NASA-CR-177930
19850027330

NASA Contractor Report 177930

AUTOSIM
AN AUTOMATED REPETITIVE RUN SOFTWARE TESTING TOOL

Janet R. Dunham
Sam E. McBride

Software Research and Development
Center for Digital Systems Research
Research Triangle Institute
Research Triangle Park, North Carolina 27709

Contract NAS1-16489
Task Assignment No. 24
September 1985

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

NASA Contractor Report 177930

AUTOSIM
AN AUTOMATED REPETITIVE RUN SOFTWARE TESTING TOOL

Janet R. Dunham
Sam E. McBride

Software Research and Development
Center for Digital Systems Research
Research Triangle Institute
Research Triangle Park, North Carolina 27709

Contract NAS1-16489
Task Assignment No. 24
September 1985

TABLE OF CONTENTS

Page No.

LIST OF FIGURES

Page No.

LIST OF TABLES

Page No.

ACKNOWLEDGEMENT

The authors acknowledge the technical direction of John Pierce during the design and development of the AUTOSIM software. We also acknowledge the attention given to this project by G. Earle Migneault of NASA-Langley Research Center and in particular his recognition of our need to develop this software. Sam E. McBride coded and tested the AUTOSIM software.

1. INTRODUCTION

1.1. Background

Digital computers are having an increasingly important role in process control applications, particularly those in which human life may be endangered such as space vehicle and avionic systems, C^3 I systems, and medical life-support systems. These systems have stringent reliability requirements as system failure is potentially life-threatening. An example is the working figure of 10^{-9} for a ten hour flight being used as a requirement for system failure probability by NASA-Langley Research Center (NASA-LaRC).[1]

The need for predicting the reliability of the software components of these life-critical systems has become more apparent with the increasing functionality being ascribed to them. The absence of a credible reliability prediction methodology for highly reliable software makes the system reliability analyses chimerical at best. The development of this methodology is hindered by our lack of knowledge about the underlying nature of the failure process for embedded real-time control software. This lack of knowledge contributes to our inability to completely eradicate or tolerate faults and to our lack of confidence in the extent to which we have approximated the goal of zero defects. As evidenced by the first well-publicized Space Shuttle software bug, the failure of the initialization logic in J. Garman's words resulted from a "very small, very improbable, very intricate, and a very old mistake."[2]

This bug typifies the rare and convoluted combination of events which cause carefully developed software to fail. It is this type of residual fault which surfaces infrequently causing a rare event or small probability failure. To detect these faults requires an order of magnitude longer time under test than the target mean time to failure.[3]

To contribute to the development of a sound statistical methodology for estimating software reliability, radar tracking software was tested using the repetitive run approach for fault rate estimation and n-version programming for error detection.[4] Four implementations of a Launch Interceptor Condition (LIC) module for a radar tracking application have been subjected to a long time under test with over 15,000,000 test cases being executed. To expedite the collection of repetitive run failure data, the AUTOSIM tool was developed.

1.2. Repetitive Run Testing

Repetitive run testing was first advocated by Nagel and Skrivan of Boeing Computer Services.[5] The repetitive run test approach provides information about the probabilistic impact of detected software faults on subsequent fault detection. It involves repetitively executing a software program using different sets of test cases from its initial state or design stage (usually the code version at the end of acceptance testing) through to the detection and correction of m faults. A code version is an instantiation of an implementation of the code under test. During the repetitive runs, the sequence of program fixes result in several instantiations or versions of the code.

Repetitive run testing provides "better" estimates of the individual fault rates. On subsequent runs or replications, the testing begins again with the initial version of the

code under test, and the faults are corrected again. Replications continue until enough observations have been collected to achieve the desired level of statistical accuracy for estimating the program failure rates. Since the input stream of test data differs for each replication, the order in which faults are diagnosed and the correction applied also differs for each replication.

1.3. N-Version Error Detection

To detect output errors from the radar tracking software, the technique of n-version programming was employed. N-version programming, first widely publicized by Avizienis[6] and further discussed by Anderson and Lee,[7] involves n programmers independently coding the problem from the same specification. A software tool, the N-VERSION CONTROLLER,[8,9] which controls the execution of each of the n independently coded implementations of the code under test and signals discrepancies between the n output vectors, was constructed for this purpose. The codes under test are the software modules being tested for their reliability. We refer to each code under test as an Application Task$_i$ (AT_i) where i=1,...,n. Using n-version programming for error detection avoids reliance on a standard to determine output correctness.

1.4. The Need for AUTOSIM

The need for a system to automate the repetitive run test process became apparent during the testing of the radar tracking software once we observed that diagnosing the fault and correcting the code under test was a time consuming, error prone process. Performing the diagnosis-correction task requires an individual with at least one year of programming experience. However, after repeating the task for several replications, it becomes very mundane and the programmer begins to perform this task by rote making it a tedious, error prone process. Moreover, the timely completion of the diagnosis-correction task is contingent upon the availability of the programmer. Since the programs fail at random points in time, the speed with which the data is collected is inhibited by the programmer's availability. For these reasons, we decided to develop the AUTOSIM system which performs repetitive run testing with a minimum of human intervention.

1.5. Definition of Terms Related to AUTOSIM

Understanding nomenclature throughout the AUTOSIM report is essential to understanding the purpose and functionality of the AUTOSIM tool. We use the following terms throughout the report. The definitions of failure, error, and fault are consistent with those defined in "Fault Tolerance by Design Diversity: Concepts and Experiments."[10]

CODES UNDER TEST — The software modules being tested for their reliability which are referred to as AT_i for Application Task$_i$.

CONDITIONS MET MATRIX (CMM) — The principal output from the Launch Interceptor Condition (LIC) Application Tasks.

CMS — VAX 11/780 Code Management System

DESIGN STAGE — Versions of the code under test during repetitive run testing

ERROR — The discrepancies noted (i.e., the incorrect element(s) of the output variables)

FAILURE — THE N-VERSION CONTROLLER signals a discrepancy in the output variables of the launch interceptor condition software. For this problem, an application task fails when it incorrectly disagrees with any of the other application tasks or the extensively tested version.

FIX/FAULT — A FIX is the minimum code change required to correct a single error. The FAULT is the conceptual flaw in the software which is corrected by a fix and is the cause of the error. For simplicity, we consider the fault to be defined by the fix and use the term fault in lieu of the term fix throughout the report although we recognize the real fault is not uniquely defined.

IMPLEMENTATION — An independently coded version of the same functional specification (i.e., one of the n-versions of the code under test)

LAUNCH — Critical output variable from the LIC code under test.

LAUNCH INTERCEPTOR CONDITION (LIC) Problem — A radar tracking application which is the first AUTOSIM test specimen.

N-VERSION CONTROLLER — Testbed used for testing n versions of an AT in parallel.

N-VERSION CONTROLLER INTERFACE — Tool used for monitoring the execution of the N-VERSION CONTROLLER and viewing stored test results.

REPLICATION/REPETITIVE RUN — Repeats of the repetitive run test beginning with the initial version of the code under test and using a different set of test cases. A REPLICATION is sometimes referred to as a REPETITIVE RUN.

VERSION — An instantiation of an implementation of the code under test. During the software fault diagnosis-correction process, the program fixes result in several instantiations or versions of the code.

2.0. OVERVIEW OF THE AUTOSIM TOOL

2.1. Design Goals

The automation of the error diagnosis-correction tasks requires a knowledge base that contains information about the documented software faults and the associated code fixes, as well as knowledge about the fault diagnosis process. These requirements categorize the AUTOSIM system as an "expert" system which replaces a low level of programming expertise.

In designing the AUTOSIM system, we pursued the following goals:

- separate concerns by maintaining a clean interface with the N-VERSION CONTROLLER and isolating the information specific to the code under test
- make the system general by storing the information specific to the code under test and the error detection state information in generalized data structures
- minimize the complexity of the fault diagnostic task by defining error classes, which are handled separately, and developing an efficient algorithm to minimize the fault diagnostic time.

Accomplishment of the first goal is important if AUTOSIM is to be used with test tools other than the N-VERSION CONTROLLER and with different application code under test. The second goal minimizes the amount of overhead required when modifying the system to test code from other applications and other error detection tools. The existence of non-unique 1-to-n mappings of faults to errors necessitates achievement of the third goal.

2.2. The AUTOSIM Algorithm

AUTOSIM performs two principal functions: fault diagnosis and fault correction. Fault diagnosis entails identifying which fixes to apply to the failed code based on the information contained in the N-VERSION CONTROLLER state vector. Critical to this identification is the implementation of an efficient algorithm for rapid identification and testing of the candidate fixes. Fault correction entails installing the appropriate fix and resuming the n-version testing. The logic for fault correction is similar to the logic for the testing of candidate fixes. Figure 1 provides a pseudo-code description of the AUTOSIM algorithm.

```
ON STOP OF TEST
   FETCH TEST STATE
   DETERMINE FAILED CODE(S)
   FOR EACH FAILED CODE
      TEST CODE VERSION WITH ALL KNOWN FAULTS CORRECTED
      ON FAILURE SIGNAL USER

      DETERMINE ERROR(S)

      FOR EACH ERROR
         DETERMINE ERROR CLASS

         IF CLASS=ADDRESS
            DETERMINE CANDIDATE ADDRESS FIXES
            FOR EACH ADDRESS FIX
               INSTALL AND TEST ADDRESS FIX
               ON SUCCESS GET NEXT ERROR
            CALL USER

         IF CLASS=ABEND
            DETERMINE CANDIDATE ABEND FIXES
            FOR EACH ABEND FIX
               INSTALL AND TEST ABEND FIX
               ON SUCCESS GET NEXT ERROR
            CALL USER

         IF CLASS=ERROR
            DETERMINE CANDIDATE ERROR FIXES
            FOR EACH ERROR FIX
               INSTALL AND TEST ERROR FIX
               ON SUCCESS GET NEXT ERROR
            CALL USER
   RESUME TEST
```

Figure 1.
Pseudo-Code Description of
Fault Diagnosis-Correction Algorithm

2.3. The AUTOSIM Design

Figure 2 provides a structural view of the AUTOSIM tool. The diagram shows the quasi-static data structures, which remain relatively constant during testing, and the dynamic data structures, which are updated by either the AUTOSIM software or the N-VERSION CONTROLLER software.

The contents of the quasi-static data structures depend on the code under test. The overwrite, abend, and output error maps contain information on which code fixes are associated with different types of faults. These structures are quasi-static because they are updated only when a new fault is identified. This identification results from the human intervention required when AUTOSIM fails to diagnose the fault (i.e., there are no valid entries in the error maps described in Section 4.2 of this report).

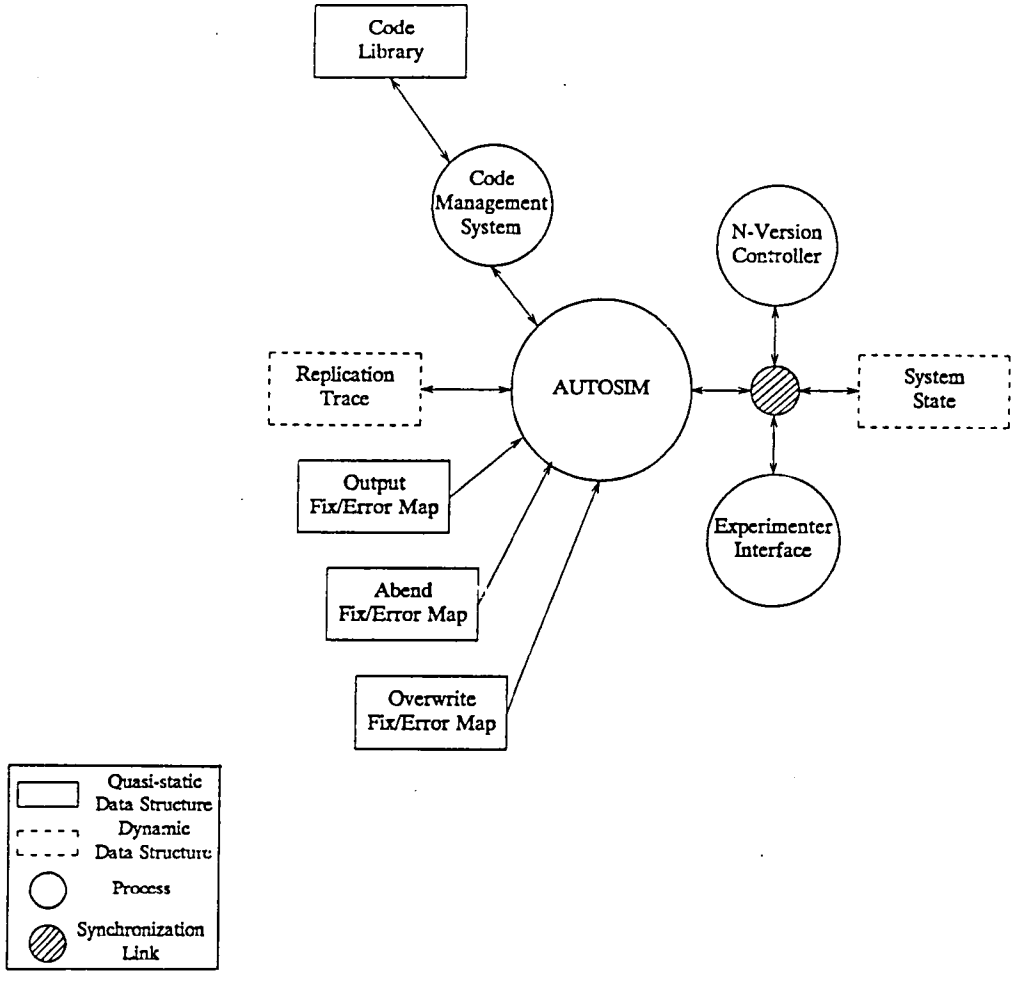


Figure 2.
A Structural View of the AUTOSIM Tool

3. IMPLEMENTATION OF THE AUTOSIM TOOL

3.1. Control Flow

Figure 3 provides a high level description of the AUTOSIM control flow and the associated data files. The three major functional modules which comprise AUTOSIM are FIXID, MEAS_IMP, and RESTORE. The VMS command procedures invoked by AUTOSIM (also depicted in Figure 3) are SIMBATCH.COM, FIXAPP.COM, and VTEST.COM. There are 7 major types of files used by AUTOSIM. These files are ATGEN.DAT, EXECUTION.DAT, iABEND.DAT, iCLOBBER.DAT, iCMM.DAT, iLAUNCH.DAT, and ERRORS.DAT.

The following sections describe the AUTOSIM functions, files, and command procedures.

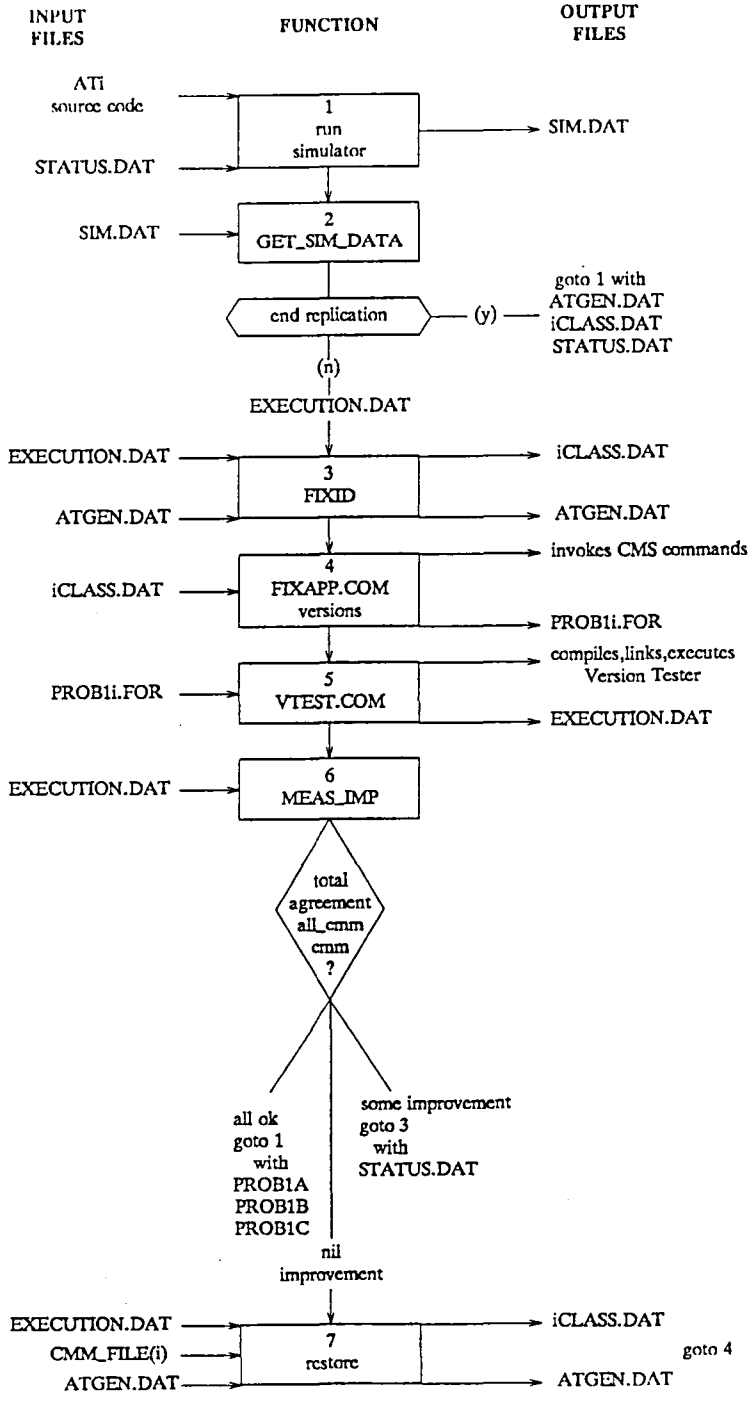


Figure 3.
AUTOSIM Global Control Flow

3.2. Descriptions of AUTOSIM Functions

The following alphabetical list provides brief descriptions of all AUTOSIM functions. Schematic logic diagrams which describe the interfaces between these functions are provided in Appendix A.

ABEND_ERROR — Finds the CMS generation for AT_i abend error.

APPEND — Adds fixes into the fix-list for AT_i.

AUTOSIM — Keeps the simulator running by identifying errors and installing fixes in the individual applications tasks.

B_SEARCH — Finds the fixes associated with the last element generations that were superseded into AT_i's CMS class.

CLOBBER_ERROR — Finds the CMS element generation to fix AT_i overwrite.

CLOSE_FILES — Closes AUTOSIM data files before spawning a subprocess.

CMM_ERROR — Finds the CMS element generation with a fix for a particular AT_i output error.

ECHO_ERROR — Writes AUTOSIM error diagnostics to file AUTOERR.DAT.

FIXID — Identifies where an application task failure occurred and determines which element generation to supersede in the CMS class.

GET_SIM_DATA — Gets a snapshot of global memory and prepares for new design stage or new replication.

ID_FIXES — Identifies the fixes associated with any CMS element generation.

LAUNCH_ERROR — Finds the CMS element generation for AT_i launch error.

MAKE_ICLASS — Builds the file which communicates with the DCL command procedure FIXAPP.COM. The file iCLASS.DAT tells FIXAPP.COM what elements are to be superseded by new generations into the appropriate CMS class.

MEASURE_IMP — Measures the effect that the last fix had on a particular AT_i's CMM.

OPEN_FILES — Opens the global input & output files.

PREP_GLO — Prepares global memory for either a new design stage or a new replication and builds the command procedure FORLINREP.COM that will compile the modified application tasks. Places the object module in the library SIMDISK:PROBILIB.OLB and links a new executable image of the N-VERSION CONTROLLER.

READ_ATGEN — Reads AT_i records from ATGEN.DAT.

READ_CMMFILE — Reads the available fixes contained in CMS element generations from the iCMM.DAT file for the AT_i under consideration.

READ_EXECUTION — Reads the file ASIM_DATA: which serves as the communication link between AUTOSIM and the spawned subprocess executing VTEST.COM.

READ_SIM.DATA — Reads last record written to SIM.DAT.

READ_TRACEBACK — Reads the AT_i traceback from ERRORS.DAT after an abend has occurred.

RESTORE — This module is executed in the event that a lame element generation was superseded into the AT_i's CMS library, i.e., the element generation that was last inserted did not contain the necessary fix. Restores the last element generation that was superseded into the AT_i's class to its previous generation and find the next element generation to install.

SET_MASK — Sets the fix mask which is displayed by the INTERFACE to the N-VERSION CONTROLLER and allows the operator to know what fixes are currently installed.

SPY_GLOBAL — Accesses global memory and returns latest sequence number.

VTEST — Tests an individual AT with the contents of the N-VERSION CONTROLLER INPUT.DAT file and compares the CMM and LAUNCH output to that computed by the golden AT.

VOID_FIX_LIST — Initializes the fix list to zero.

WRITE_ATGEN — Updates application task records in the file ASIM_DATA:ATGEN.DAT. ATGEN.DAT is a direct access file containing

information about the CMS class for each application task.

WRITE_EXECUTION — Writes to the file ASIM_DATA:EXECUTION.DAT.

ZERO_SUPER_ELE — Initializes the supersede section (the second group of N_ATS records) in the file ATGEN.DAT. This is to ensure that only the next element/generation elected to fix an error will be superseded into the appropriate AT_i class in the CMS library.

3.3. Descriptions of AUTOSIM Files

The following alphabetical list provides descriptions of the AUTOSIM data files. More complete descriptions can be found in Appendix B.

ATGEN.DAT — Contains the trace of CMS element generations that were installed, indicates those that are presently installed, and those which can be superseded. ATGEN.DAT is accessed by the functions READ_ATGEN and WRITE_ATGEN.

EXECUTION.DAT — Contains information pertaining to results of executing the code in error with the diagnosed fix. EXECUTION.DAT is accessed by the functions READ_EXECUTION and WRITE_EXECUTION.

iABEND.DAT — Contains a history of the abend failures which have been documented for each AT_i during the execution of the Launch Interceptor Code. The function ABEND_ERROR accesses iABEND.DAT.

iCLOBBER.DAT — Contains a history of the overwrite failures which have been documented for each AT_i during the initial execution of the Launch Interceptor Code. The function CLOBBER_ERROR accesses iCLOBBER.DAT.

iCMM.DAT — Contain the fault-error maps for the documented faults. The function CMM_ERROR accesses iCLOBBER.DAT.

iLAUNCH.DAT — are used when an AT_i fails and the failure does not show up as an abend, overwrite or as disagreement in the Conditions met matrix. The function LAUNCH_ERROR accesses iLAUNCH.DAT.

ERRORS.DAT — Contains all system trackbacks when a routine in the N-VERSION CONTROLLER ends abnormally. The function READ_TRACEBACK accesses ERRORS.DAT.

3.4. AUTOSIM Command Procedures

The following alphabetical list provides brief descriptions of the AUTOSIM command procedures. Listings of the command procedures can be found in Appendix C.

AUTOSIM.COM — Submitted as a batch job to start the execution of AUTOSIM.

FIXAPP.COM — Creates working versions of each of the AT's.

FORLINREP.COM — A dynamically created command procedure which compiles the application tasks, places them in PROB1LIB.OLB (the problem 1 object library) and links the N-VERSION CONTROLLER.

SIMBATCH.COM — Submitted as a batch job to start the execution of the N-VERSION CONTROLLER.

VTEST.COM — Tests fixed version of AT with error-invoking input case.

4. AUTOSIM DEPENDENCIES

4.1. Management of the Code under Test

The VAX11/780 Code Management System (CMS) is used to manage the library containing the code under test.[11] CMS is a program library system for software development and maintenance which operates as an online librarian by keeping track of the source code files. Each CMS library contains elements, generations, and classes.

A CMS element is the basic structural unit in a code library. An element consists of one or more ASCII files that represent a meaningful unit. An element is created and named when a file (or files) is transferred from a working directory to the CMS library via the CMS CREATE ELEMENT command. The AUTOSIM *elements* are functionally defined modules of the each implementation of the code under test.

A CMS element generation represents a phase in the development of that element. When an element is created and placed in the CMS library for the first time, CMS creates generation one of that element. Each time the element is reserved, modified, and replaced in the CMS library, a new generation is created. The AUTOSIM *generations* are the functionally defined modules with different fixes applied.

A CMS class is a set of generations of different elements. A class is established to define a set of generations that make up the whole of part of a software system at a specific stage of development. The AUTOSIM *classes* pertain to the different codes under test, i.e., to the different AT_i .

Tables 1, 2, and 3 depict the organization of the code library for the testing of the Launch Interceptor Condition Software. The module names are the actual names of the different software modules given by the programmers. A number identifies each fault in an AT_i.

In developing the AUTOSIM code library, we partitioned each AT into modules which correspond to conditions of the LIC problem and into modules which contain subroutines common to the test condition modules. Modules with no fixes were coupled with modules that had corresponding fixes (i.e., placed in the same CMS element) to save storage space. Subsequent generations of each element are the versions of the code under test with different fixes applied. CMS stores the subsequent generations of a CMS element by retaining the code differences from the first generation element. The update of the version of the code under test to correct a fault does not necessarily result in the installation of the next generation of an element. For example, installation of the fix associated with Fault 8 may be required for element A09 of AT1 prior to installation of the fix associated with Fault 7.

4.2. Code Under Test Fix-Error Maps

Fix-error maps define the relationship between code under test output errors (in this case errors in the CMM and in LAUNCH) and fixes which correct those errors. Tables 4, 5 and 6 depict the fix-error maps for the three implementations of the LIC problem

TABLE 1. CLASS DESCRIPTIONS OF AT_i

CLASS	ELEMENT	GENERATION	DESCRIPTION	
			Module Name	Faults Corrected
AT1	A01	1	main	
		2	main	12
	A02	1	cond1	
		2	cond1	9
	A03	1	cond2 cond3	
		2	cond2 cond3	10
	A04	1	cond4 cond5	
		2	cond4 cond5	2
		3	cond4 cond5	4
		4	cond4 cond5	2,4
	A05	1	cond6 cond7	
		2	cond6 cond7	3
	A06	1	cond8	
		2	cond8	6
	A07	1	cond9 cond10	
		2	cond9 cond10	11
	A08	1	cond11 cond12 cond13 cond14 cond15 main	
	A09	1	anglea	
		2	anglea	1
		3	anglea	1,7
		4	anglea	1,8
		5	anglea	1,7,8
	A010	1	dista rad	
		2	dista rad	5

TABLE 2. CLASS DESCRIPTIONS OF AT_i

CLASS	ELEMENT	GENERATION	DESCRIPTION	
			Module Name	Faults Corrected
AT2	B01	1	prob1b	
		2	prob1b	1

TABLE 3. CLASS DESCRIPTIONS OF AT_i

CLASS	ELEMENT	GENERATION	DESCRIPTION	
			Module Name	Faults Corrected
AT3	C01	1	main cond1 cond2	
		2	main cond1 cond2	3
	C02	1 2	cond3 cond3	4
	C03	1 2	cond4 cond4	5
	C04	1 2	cond5 cond5	6
	C05	1	cond6 cond7	
		2	cond6 cond7	1
		3	cond6 cond7	1,7
		4	cond6 cond7	1,16
		5	cond6 cond7	1,7,16
	C06	1 2	cond8 cond8	8
	C07	1 2	cond9 cond9	9
	C08	1 2	cond10 cond10	10
	C09	1 2	cond11 cond11	11
	C010	1 2	cond12 cond12	12
	C011	1 2 3 4	cond13 cond13 cond13 cond13	2 13 2,13
	C012	1 2	cond14 cond14	14
	C013	1 2	cond15 cond15	15
	C014	1 2	radcir radcir	18
	C015	1 2	perdis perdis	20
	C016	1 2 3	aglcos aglcos aglcos	17 17,19
	C017	1	abcisa ordnat dist errstp vfind aretri quad verin verout	

respectively. The rows in these tables identify the fault/fix number; the columns in these tables identify bit errors in the CMM, LAUNCH, abend errors and overwrite errors.

4.3. N-VERSION CONTROLLER Dependencies

The functions PREP_GLO and GET_SIM_DATA comprise the two points of communication of AUTOSIM with the N-VERSION CONTROLLER.

PREP_GLO prepares global memory for either a new design stage or a new replication, and builds the command procedure FORLINREP.COM that will compile the modified application tasks. It then places the object module in the library SIMDISK:PROBILIB.OLB and links a new executable image of the N-VERSION

TABLE 4. FIX-ERROR MAP FOR AT1

FIX	CMM OUTPUT (i)															ABENDS		OVERWRITES
ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	RAD	ANGLEA	COMMON2
1																		X
2					X													
3							X											
4					X													
5																X		
6								X					*					
7			X							X			X				X	
8																	X	
9	X																	
10			X															
11										X								
12										X								

X indicates an error in output CMM (i), an ABEND error, or an OVERWRITE error.

* indicates that FIX 6 is applied when simultaneous errors in CMM 8 and 13 occur.

TABLE 5. FIX-ERROR MAP FOR AT2

FIX	CMM OUTPUT (i)															
ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	LAUNCH
1																X

X indicates an error in output CMM (i), an ABEND error, or an OVERWRITE error.

TABLE 6. FIX-ERROR MAP FOR AT3

FIX	CMM (i)															ABENDS		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	radcir	aglcos	perdis
1							X											
2													X					
3																		
4																		
5				X														
6					X													
7							X											
8								X										
9									X									
10										X								
11											X							
12												X						
13													X					
14														X				
15															X			
16							X											
17																	X	
18																X		
19			X							X								
20																		X

X indicates an error in output CMM (i), an ABEND error, or an OVERWRITE error.

FIX 3 and FIX 4 constitute changes to perceived faults which were non-existent.

CONTROLLER.

GET_SIM_DATA, upon normal halting of the N-VERSION CONTROLLER, reads the record corresponding to the last sequence number from the SIM.DAT file. If the halt occurs during a replication, the module FIXID is invoked. If the halt occurs at the end of a replication, the command procedure FIXAPP.COM is spawned.

4.4. VMS Dependencies

The VMS command procedures VTEST.COM, SIMBATCH.COM, AUTOSIM.COM, and FORLINREP.COM are critical to the execution of AUTOSIM as described in Section 3.4. These procedures may change as a result of upgrading of the VMS DEC Command Language.[12]

5. AUTOSIM VALIDATION AND PERFORMANCE

5.1. Validation Test Results

Upon completion of unit testing of the AUTOSIM modules and white box functional testing of AUTOSIM, we tested AUTOSIM by repeating eight of the repetitive runs or replications executed during a previous conduct of the experiment.[4] The test replications selected were replications for which interacting faults were and were not observed. The testing surfaced the following problems:

- AUTOSIM execution of replication 1 identified a fix that should not have been applied.
- AUTOSIM execution of Replication 2 surfaced an error in the subroutine GENER of the N-VERSION CONTROLLER where an input (x,y) pair was not re-initialized.
- AUTOSIM execution of Replication 16 indicated that in the original replication a program version being tested had a fix to a previously corrected error inadvertently removed.

Logs describing the validation testing are provided in Appendix D.

5.2. Performance Measures

Performance of AUTOSIM was of interest for two reasons. First, we were interested in the reduction in human effort actually achieved from using the AUTOSIM tool. Replications of length 10,000 input cases executed without the use of AUTOSIM averaged two days to complete. These replications required the availability of a programmer who spent approximately four hours per day monitoring the system and installing the appropriate fixes. The AUTOSIM replications of the same length averaged six calendar hours to complete and require a maximum of one-half hour of monitoring time per day.

Second, we are interested in the efficiency of the AUTOSIM algorithm in diagnosing the appropriate fault. We are currently measuring the efficiency by number of VTEST invocations per fix required. The LOG.DAT and the CHART.DAT files contain the data required to compute this statistic.

6. USING AUTOSIM

6.1. Getting Started

AUTOSIM is executed through the N-VERSION CONTROLLER INTERFACE.[9] To start AUTOSIM, type SIM1, select option 13, and answer the prompts.

6.2. Libraries Needed

The following libraries are needed to run AUTOSIM:

PROB1LIB.TLB — which contains the N-VERSION CONTROLLER and the AT source code.

AUTO.TLB — contains the AUTOSIM source code.

MTRSRC.TLB — contains the N-VERSION INTERFACE source code.

These files presently reside on AIRLAB System 3::DRA0:[SIM.PROB1...].

6.3. AUTOSIM Error Files

The AUTOERR.DAT file contains AUTOSIM error information. Two types of information are stored in this file. The VMS System error message number when a system level error occurs and an AUTOSIM error message when AUTOSIM cannot normally execute its function. The former system level messages can be obtained by typing "exit [msg. no.]". The latter AUTOSIM error messages should be completely self-explanatory.

7.0. REFERENCES

1. J. R. Dunham and J. C. Knight, eds., "Production of Reliable Flight Crucial Software: Validation Method Research for Fault-Tolerant Avionics and Control Systems Sub-Working-Group Meeting," NASA Conference Publication 2222, NASA (1982).
2. John R. Garman, "The "Bug" Heard Round the World," *Software Engineering Notes* **6** pp. 3-10 ACM Sigsoft, (October 1981).
3. Douglas R. Miller, "Some Statistical Issues in Assurance of Very Highly Reliable Systems," *IEEE Computer Society Woirkshop on Laboratories for Reliable Systems Research* **Abstract**(April 1983).
4. J. R. Dunham and J. L. Pierce, "An Experiment In Software Reliability," NASA CR - 172553, NASA, Langley Research Center (March 1985).
5. Phyllis M. Nagel and James A. Skrivan, "Software Reliability: Repetitive Run Experimentation and Modeling," NASA CR-165836, NASA, Langley Resarch Center, Hampton, Virginia (February 1982).
6. A. Avizienis, "Fault Tolerance: The Survival Attribute of Digital Systems," *Proceedings of the IEEE* **66**(10)(October 1978).
7. T. Anderson and P. A. Lee, *Fault Tolerance Principles and Practice,* Computing Laboratory, University of Newcastle Upon Tyne, England Prentice/Hall International, London, England (1981).
8. William F. Ingogly, et al., "N-VERSION SIMULATOR INTERFACE Maintenance Guide Version 1.0," RTI Report No. 43U-2094-12b, Research Triangle Institute (October 1983).
9. William F. Ingogly, et al., "N-VERSION SIMULATOR INTERFACE User's Guide Version 1.0," RTI Report No. 43U-2094-12a, Research Triangle Institute (October 1983).
10. A. Avizienis and J.P.J. Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments," *Computer* **17** p. 69 IEEE Computer Society, (August 1984).
11. *VAX DEC-11/Code Management System Document Set,* Digital Equipment Corporation, Maynard, Massachusetts (May 1982). AA-M681A-TE
12. *VAX/VMS Document Set,* Digital Equipment Corporation, Maynard, Massachusetts (September 1984). AA-Z101A-TE
13. M. Jackson, *Principles of Program Design,* Academic Press, New York (1975).

APPENDIX A. AUTOSIM SCHEMATIC LOGIC DIAGRAMS

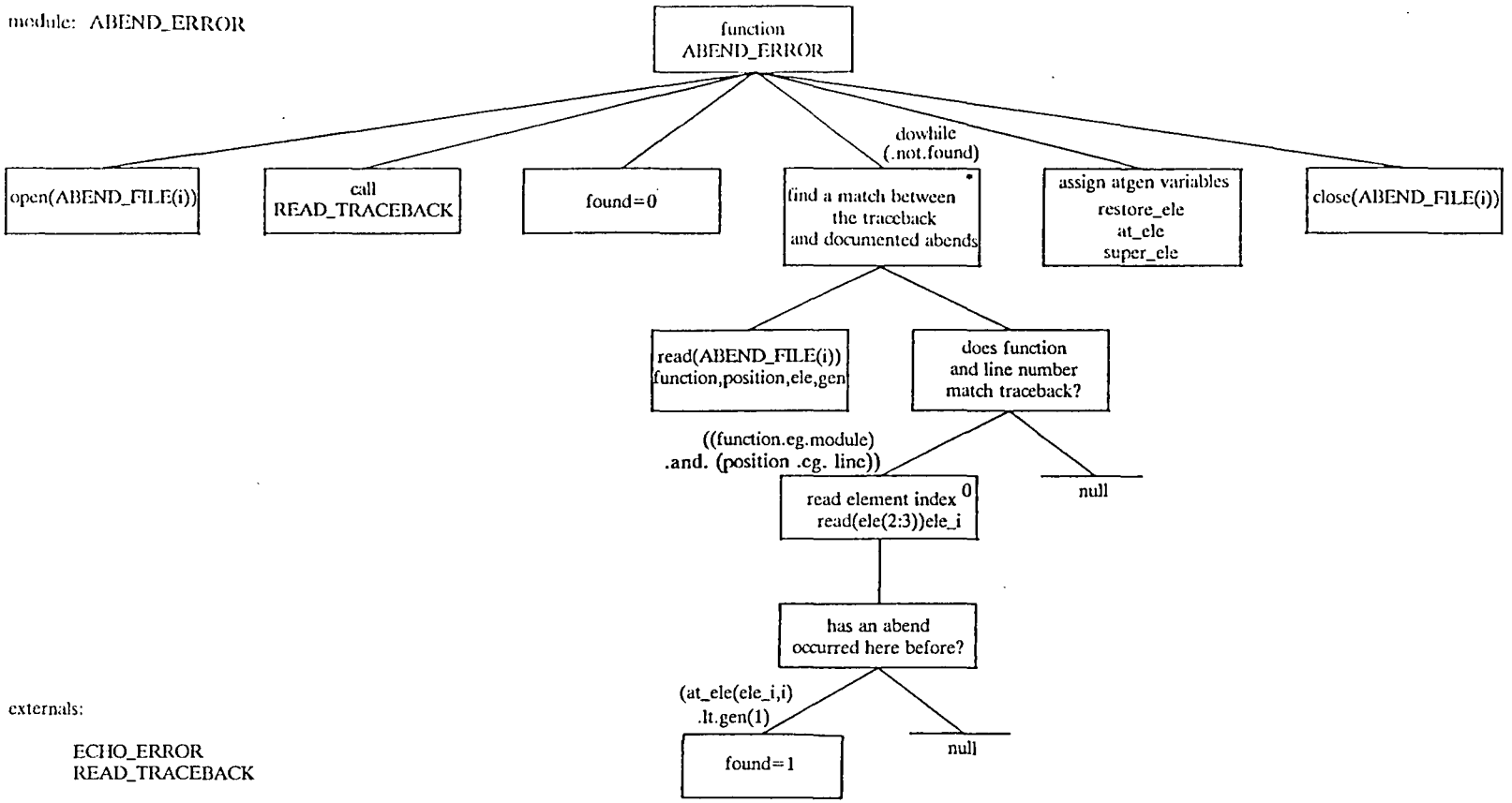
Each of the following pages corresponds to a single AUTOSIM module, or to part of an AUTOSIM module. The module name is given in the upper left hand corner of each page, and other AUTOSIM routines called by the module or module fragment are listed in the lower left part of the page. Modules are numbered sequentially in order of occurrence in the drawings, from first drawing to last, and in left-to-right order within each drawing. Arrows drawn from a subprocess box point to the number of a called module or to a continuation page for the current module. Continuation pages are numbered as fractional extensions of the module number; e.g., module n. would have its continuation pages numbered n.1, n.2, and so on, and continuation page n.m would have its sub-continuation pages numbered n.m.1, n.m.2, etc.

Each box on the diagrams represents a process or subprocess in the AUTOSIM, and lines connecting the boxes represent program control flow. Subprocess execution is from top of page to bottom, and from left to right within the same level. Conditionally executed boxes have a small circle drawn in the upper right hand corner, and the branch condition is printed above the box. Boxes that are executed iteratively have an an asterisk drawn in the upper right hand corner, and the iteration condition is printed above the process box that controls the iterated subprocess. Null branches are indicated by a horizontal line above the word "Null."

Data assignments are listed below a process box, if they're needed to understand process logic. An escape on an error condition is indicated by an arrow leading from an empty process box. Section 3.2 of this document contains an alphabetized list of all modules, with brief descriptions.

Jackson,[13] pp. 17-42 describes the notation used in these diagrams in greater detail.

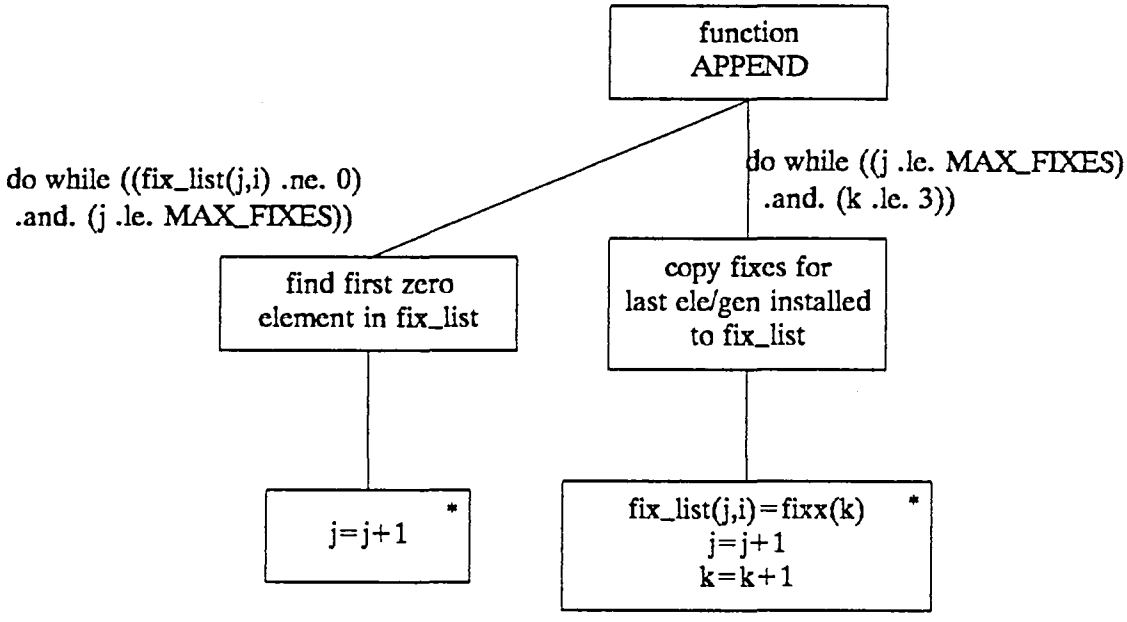
module: ABEND_ERROR



externals:

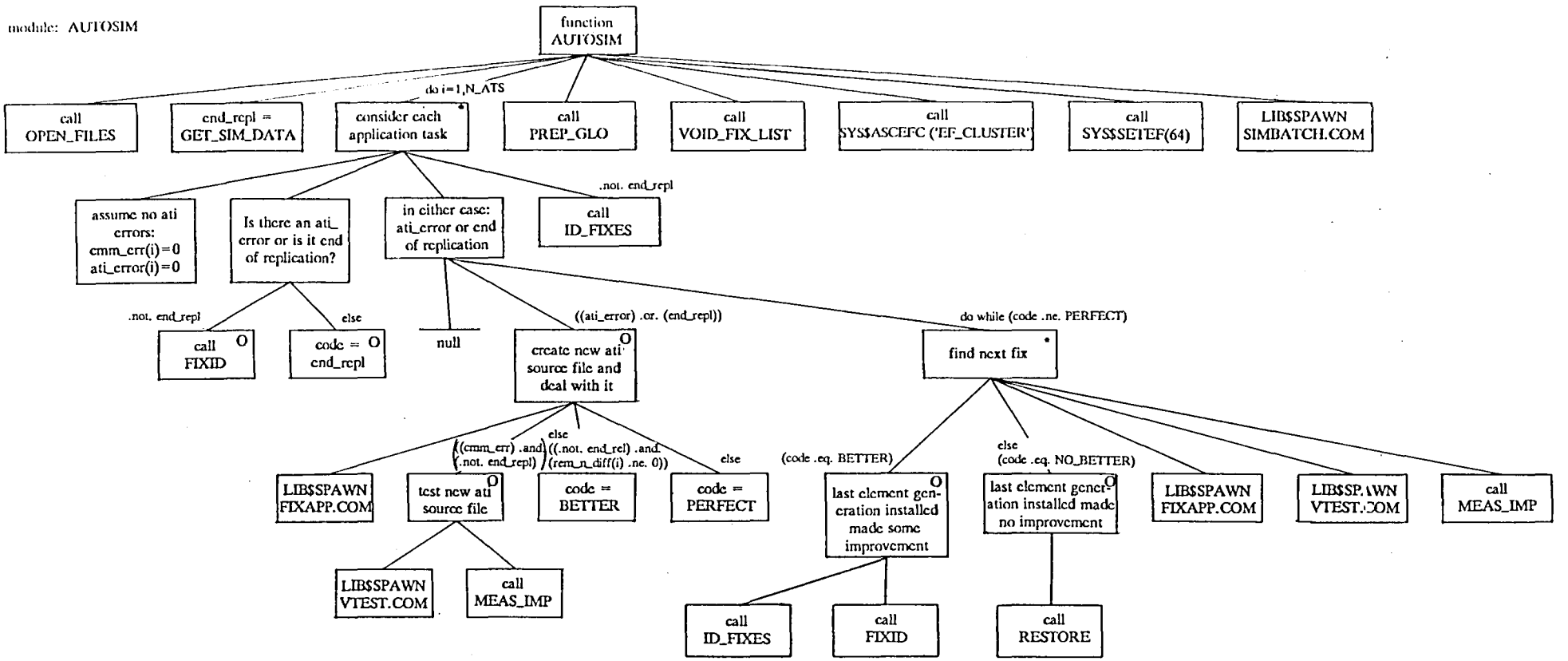
ECHO_ERROR
READ_TRACEBACK

module: APPEND



Externals:

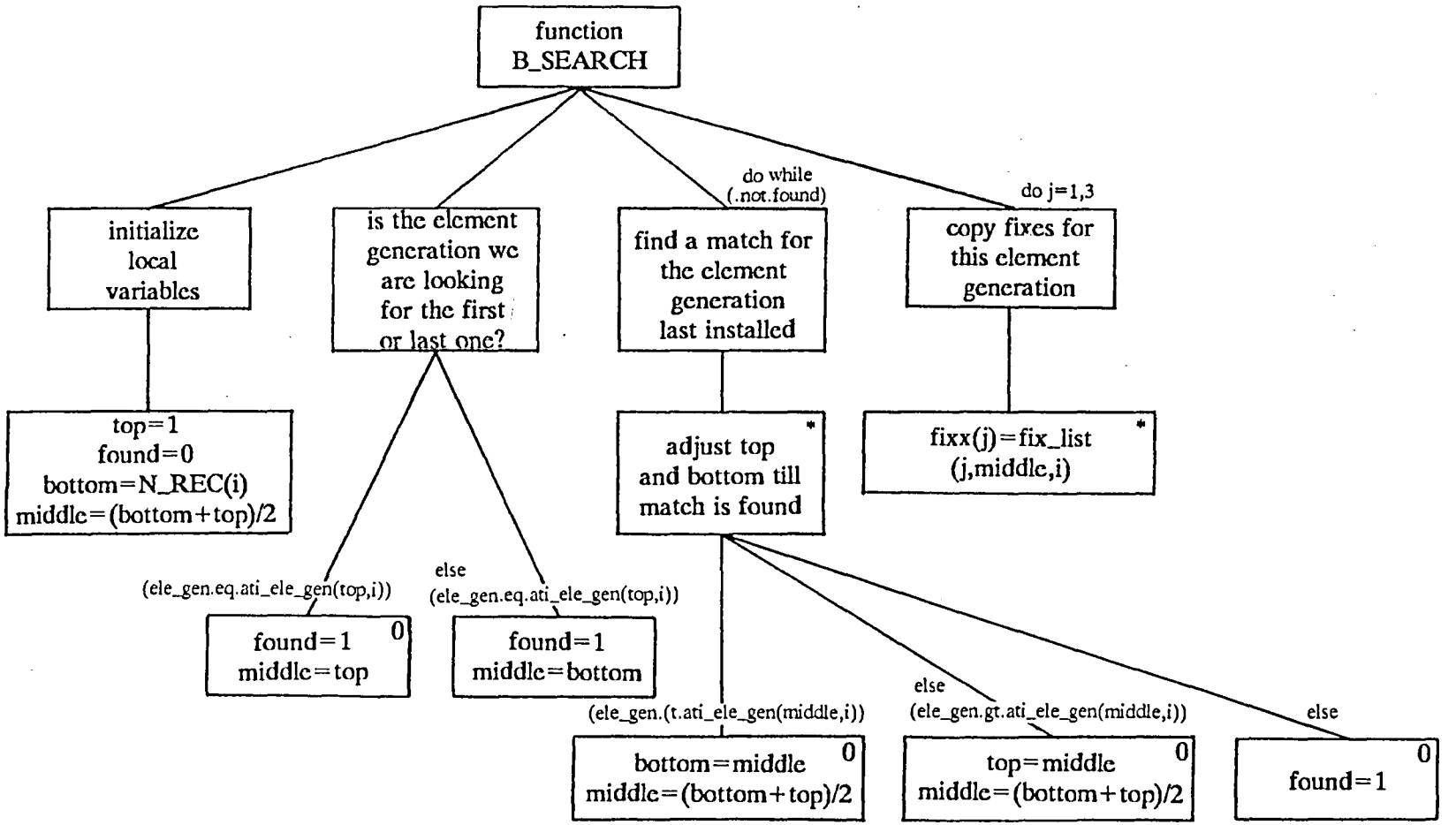
ECHO_ERROR



Externals:

SYS$ASCEFC	RESTORE
SYS$SETEF	ECHO_ERROR
OPEN_FILES	FIXAPP.COM
GET_SIM_DATA	VTEST.COM
FIXID	SIMBATCH.COM
CLOSE_FILES	PREP_GLO
MEAS_IMP	VOID_FIX_LIST
LIB$SPAWN	ID_FIXES

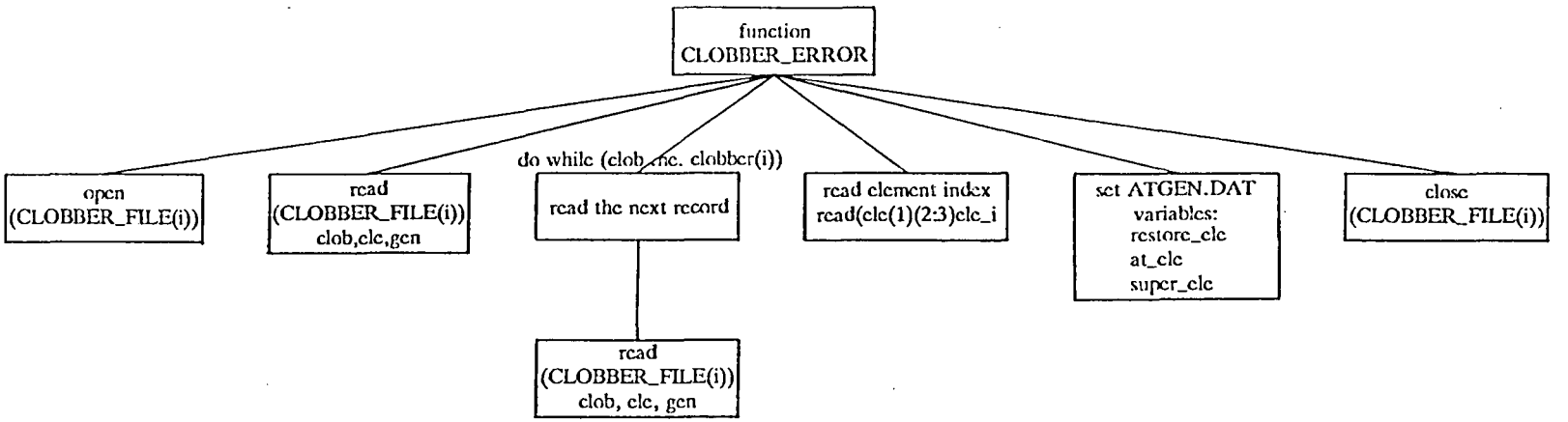
module: B_SEARCH



externals:

none

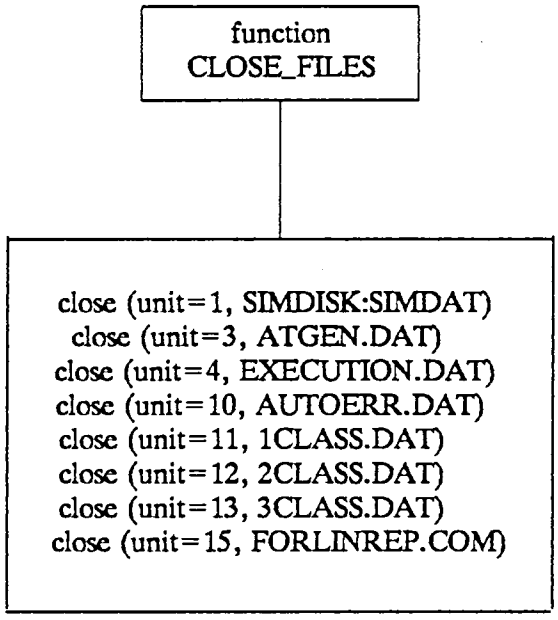
module: CLOBBER_ERROR



Externals:

ECHO_ERROR

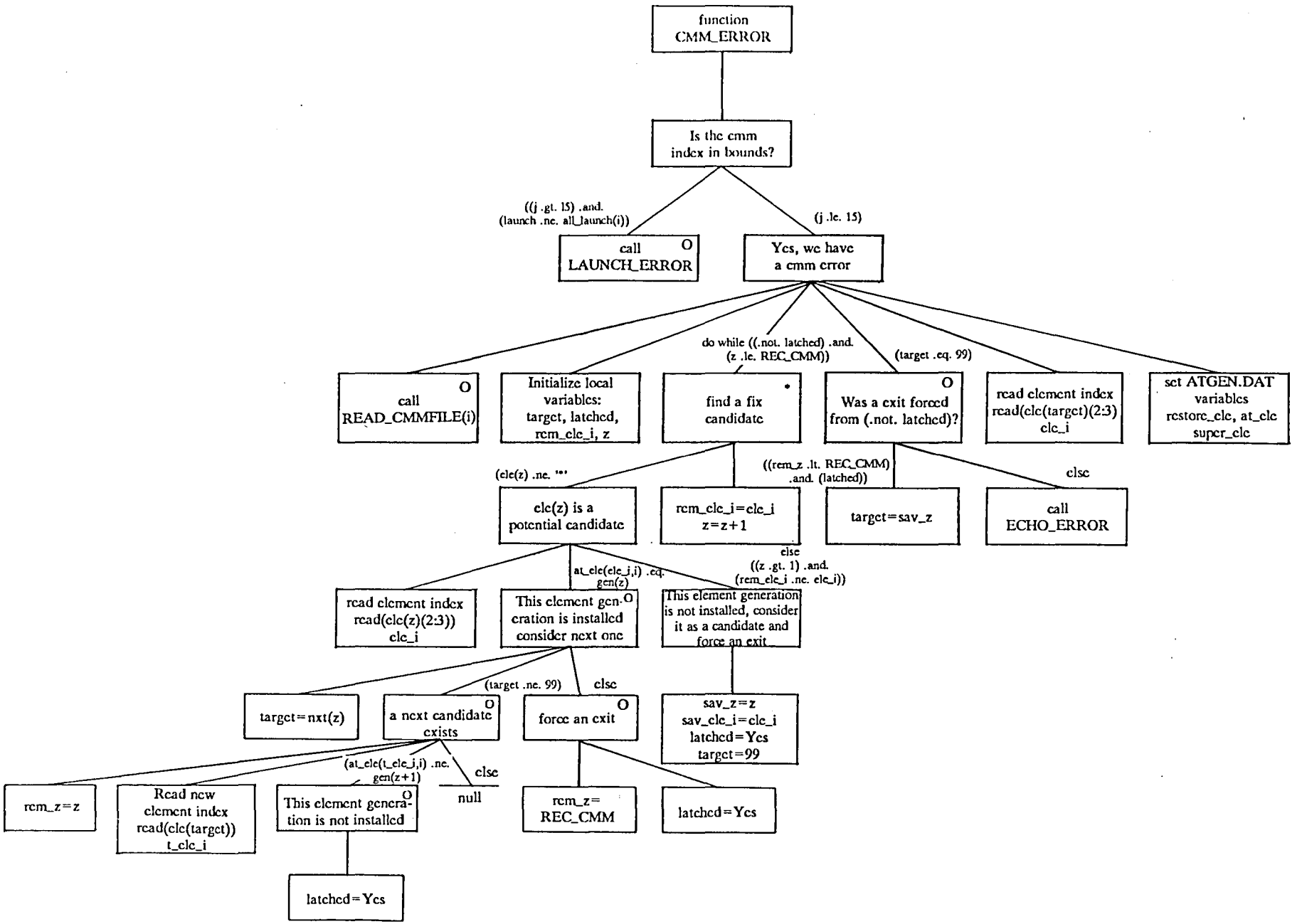
module: CLOSE_FILES



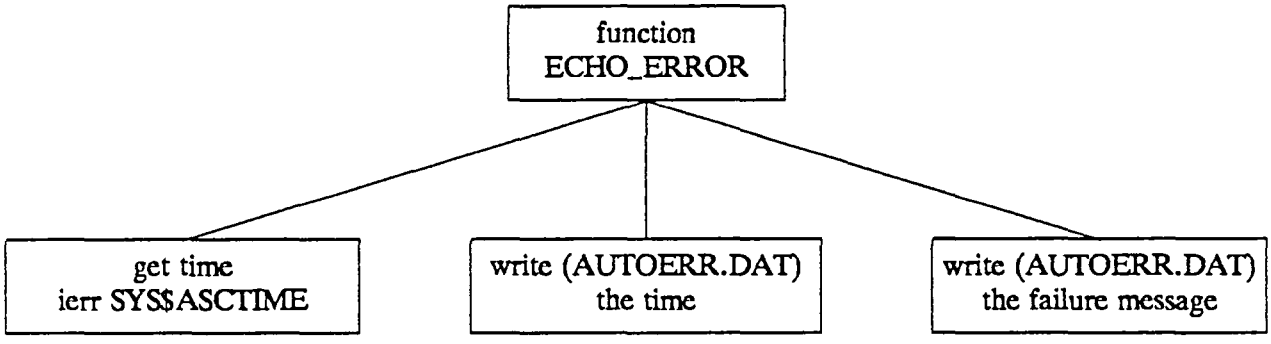
Externals:

none

module: CMM_ERROR



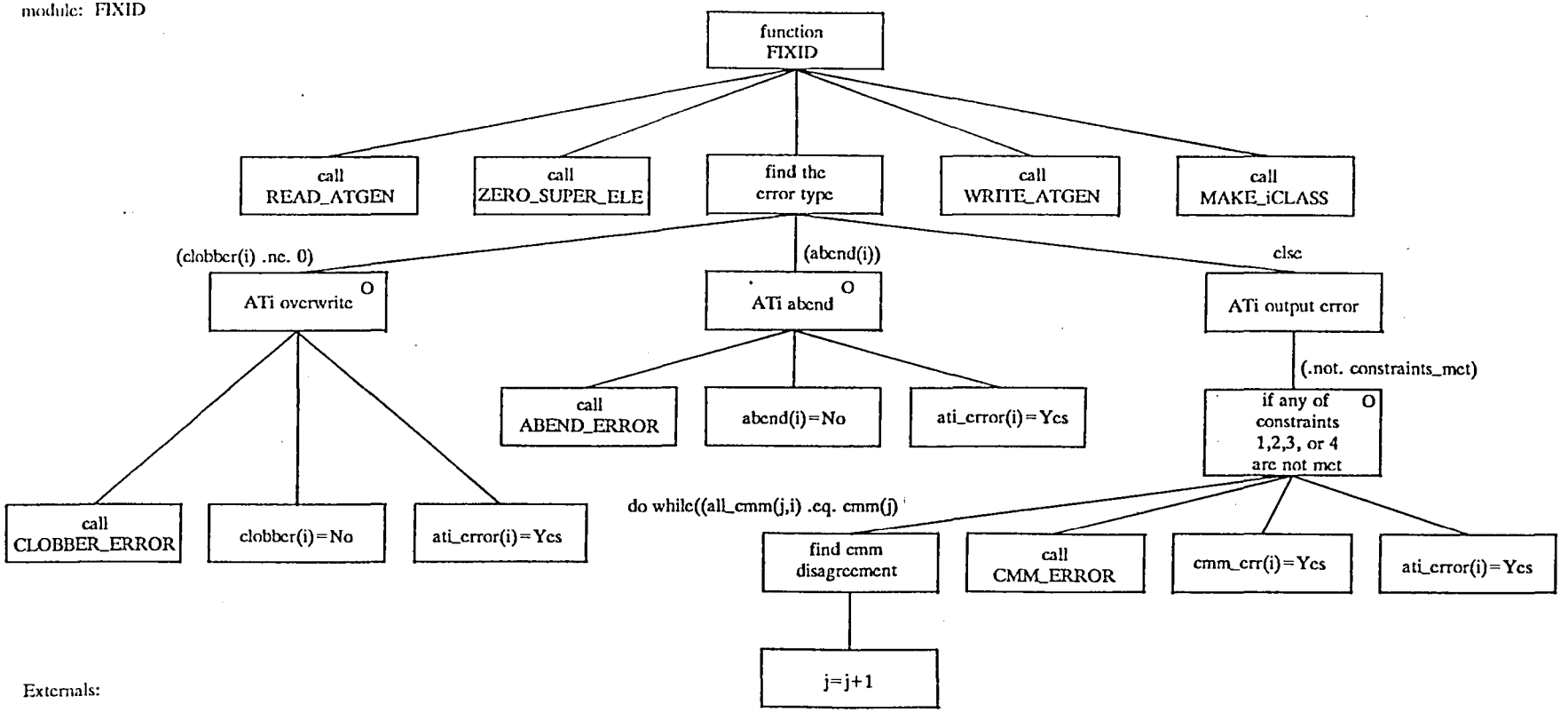
module: ECHO_ERROR



Externals:

SYS$ASCTIME

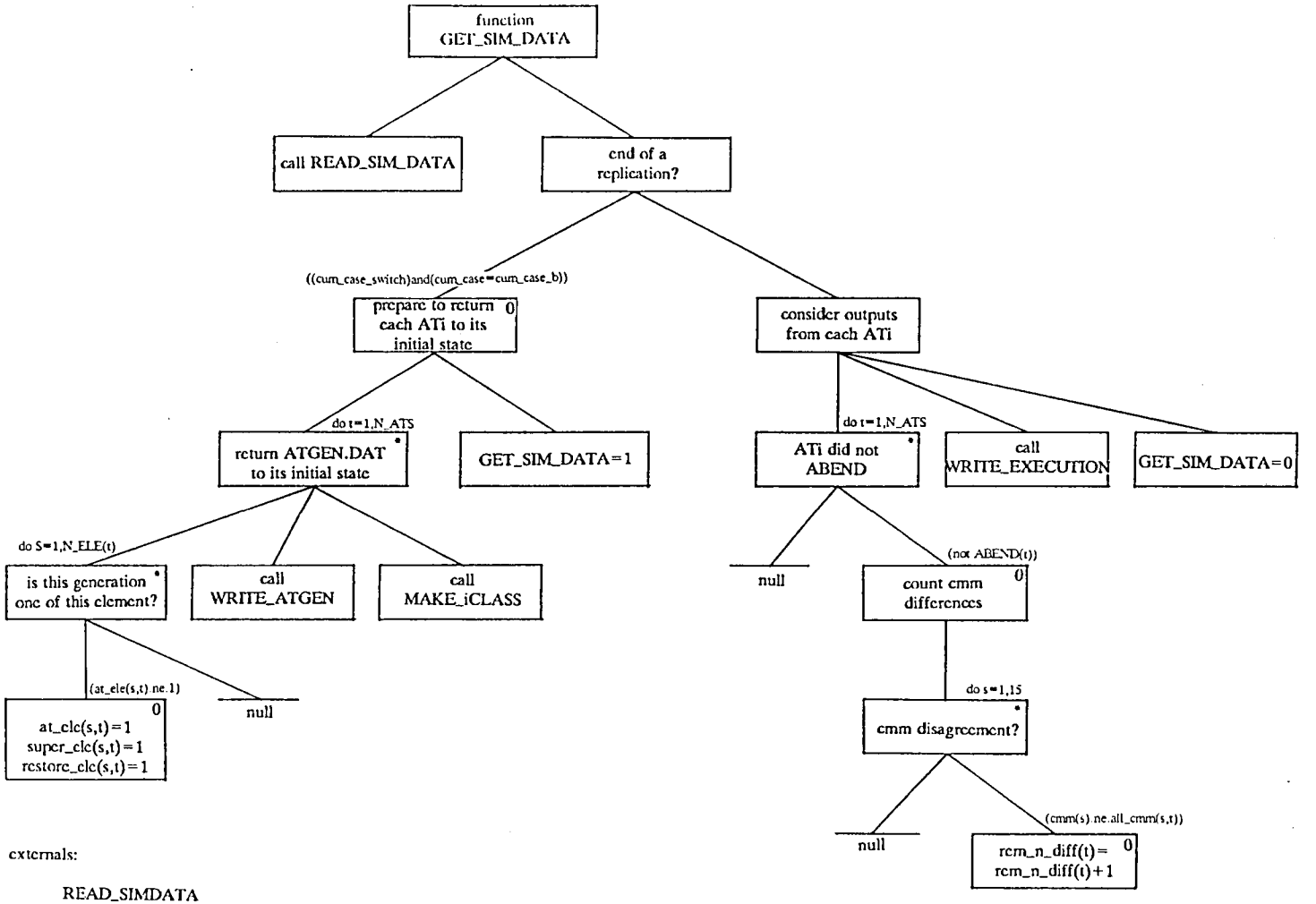
module: FIXID



Externals:

READ_ATGEN
ZERO_SUPER_ELE
CLOBBER_ERROR
ABEND_ERROR
CMM_ERROR
WRITE_ATGEN
MAKE_iCLASS

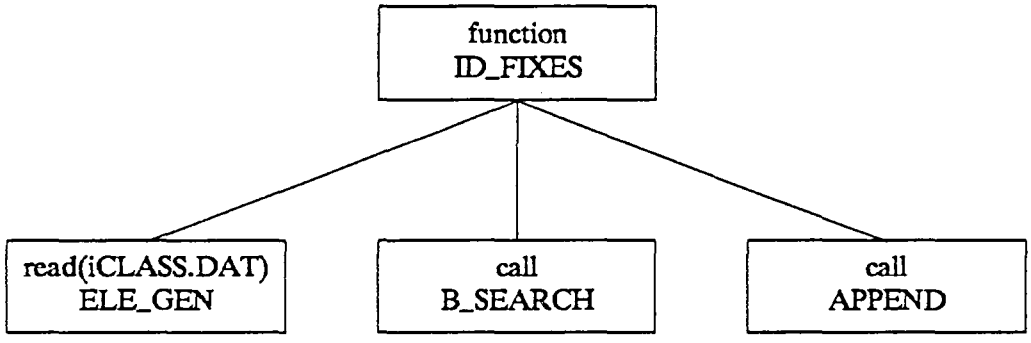
module: GET_SIM_DATA



externals:

READ_SIMDATA
READ_ATGEN
WRITE_ATGEN
MAKE_iCLASS
WRITE_EXECUTION

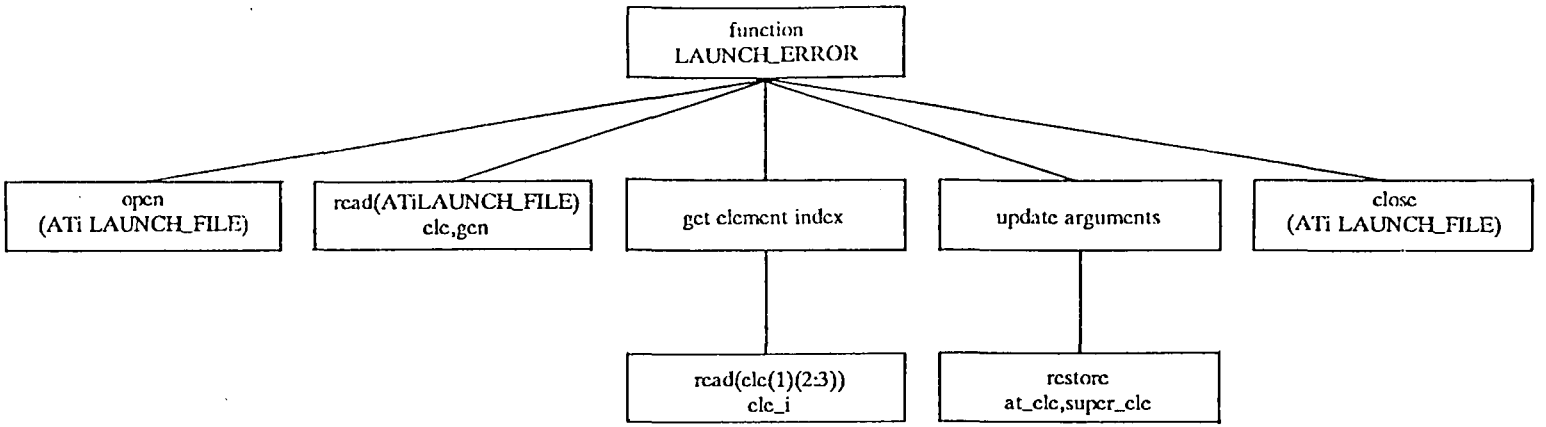
module: ID_FIXES



Externals:

B_SEARCH
APPEND

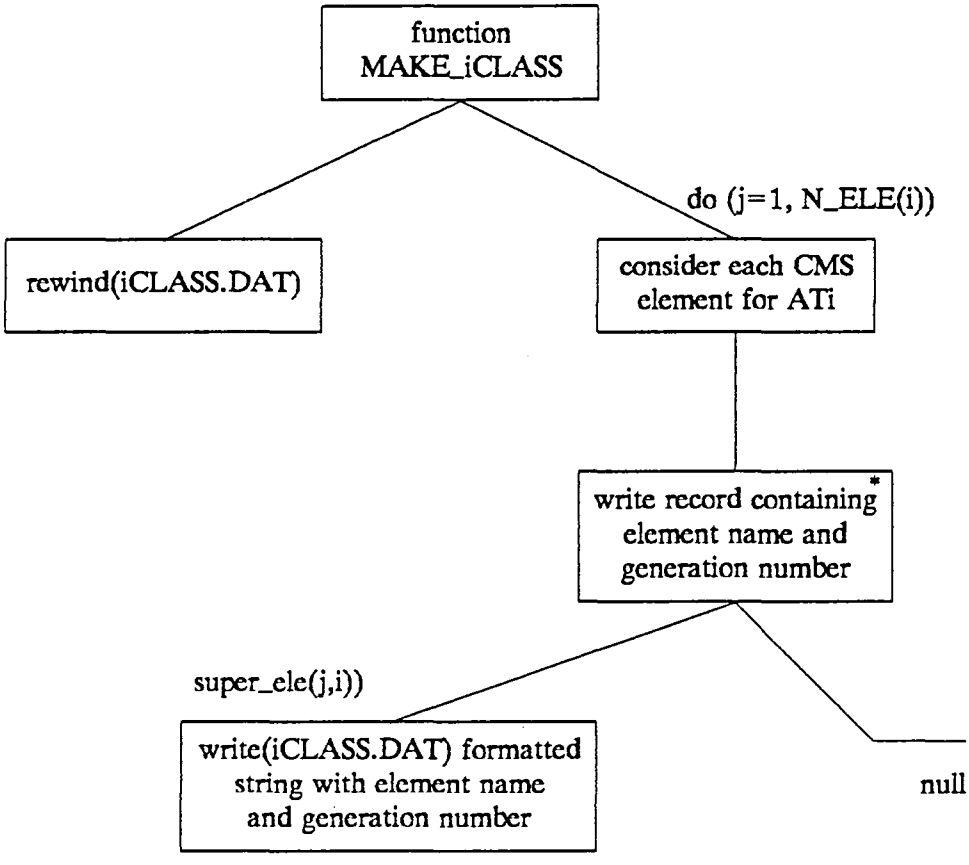
module: LAUNCH_ERROR



Externals:

ECHO_ERROR

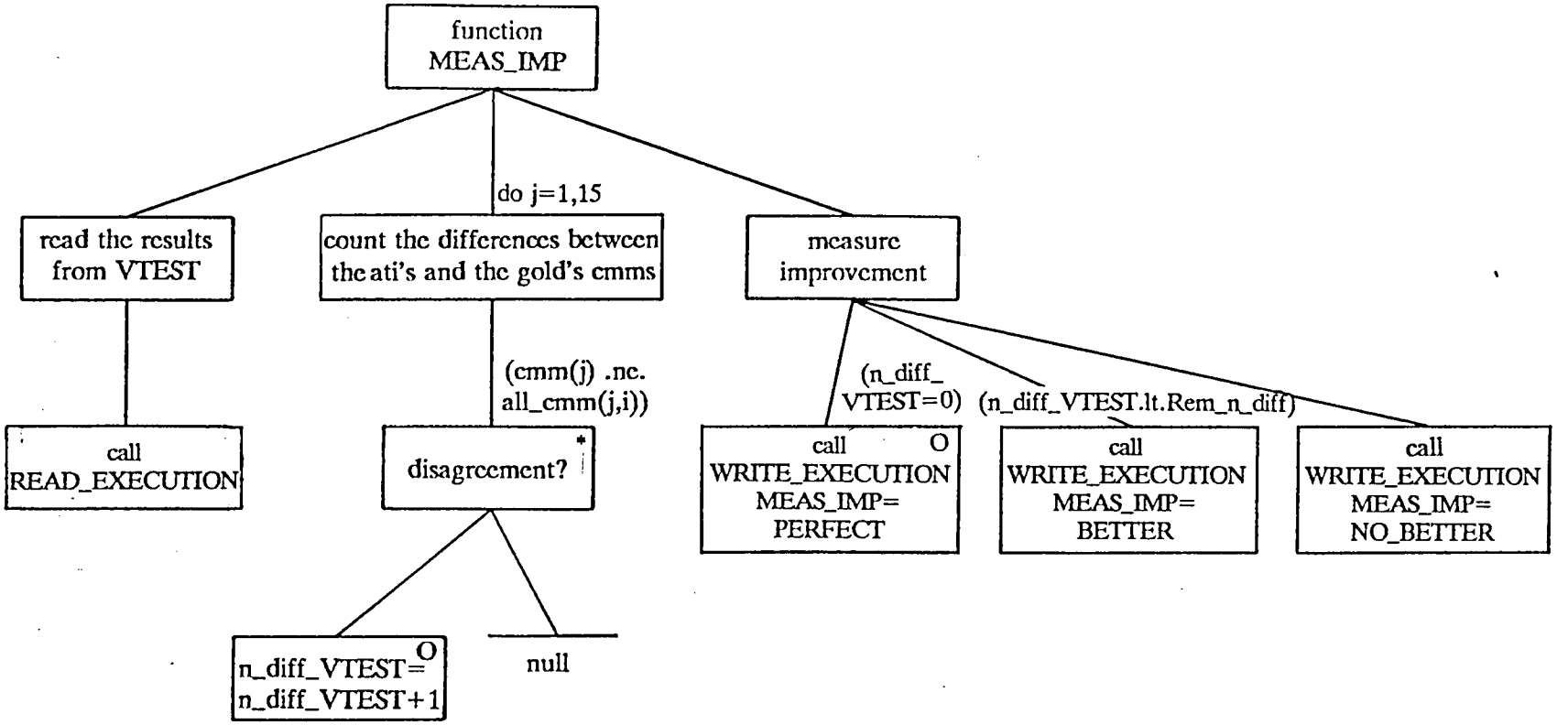
module: MAKE_iCLASS



Externals:

none

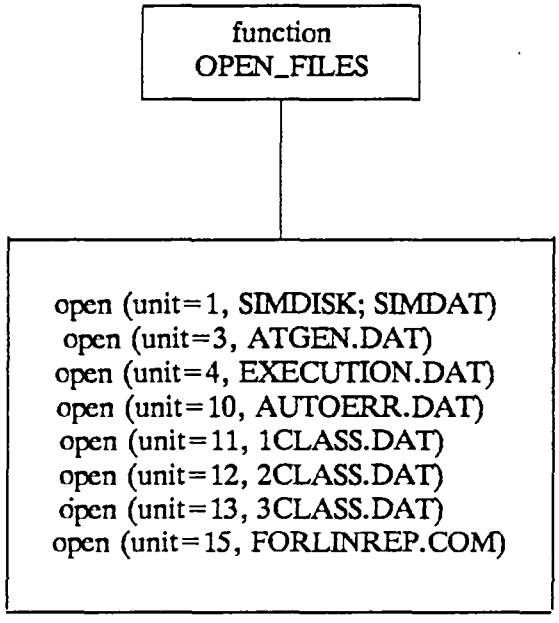
module: MEAS_IMP



Externals:

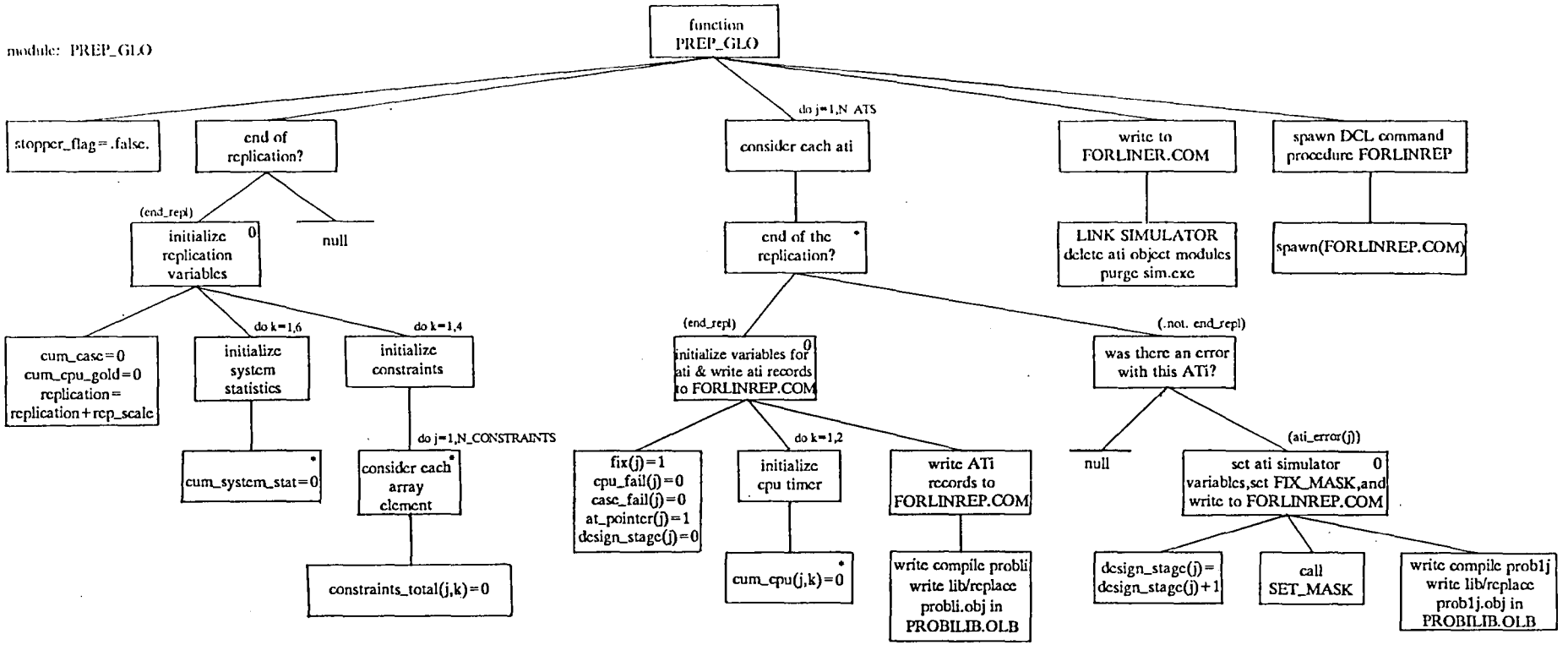
READ_EXECUTION
WRITE_EXECUTION

module: OPEN_FILES



Externals:

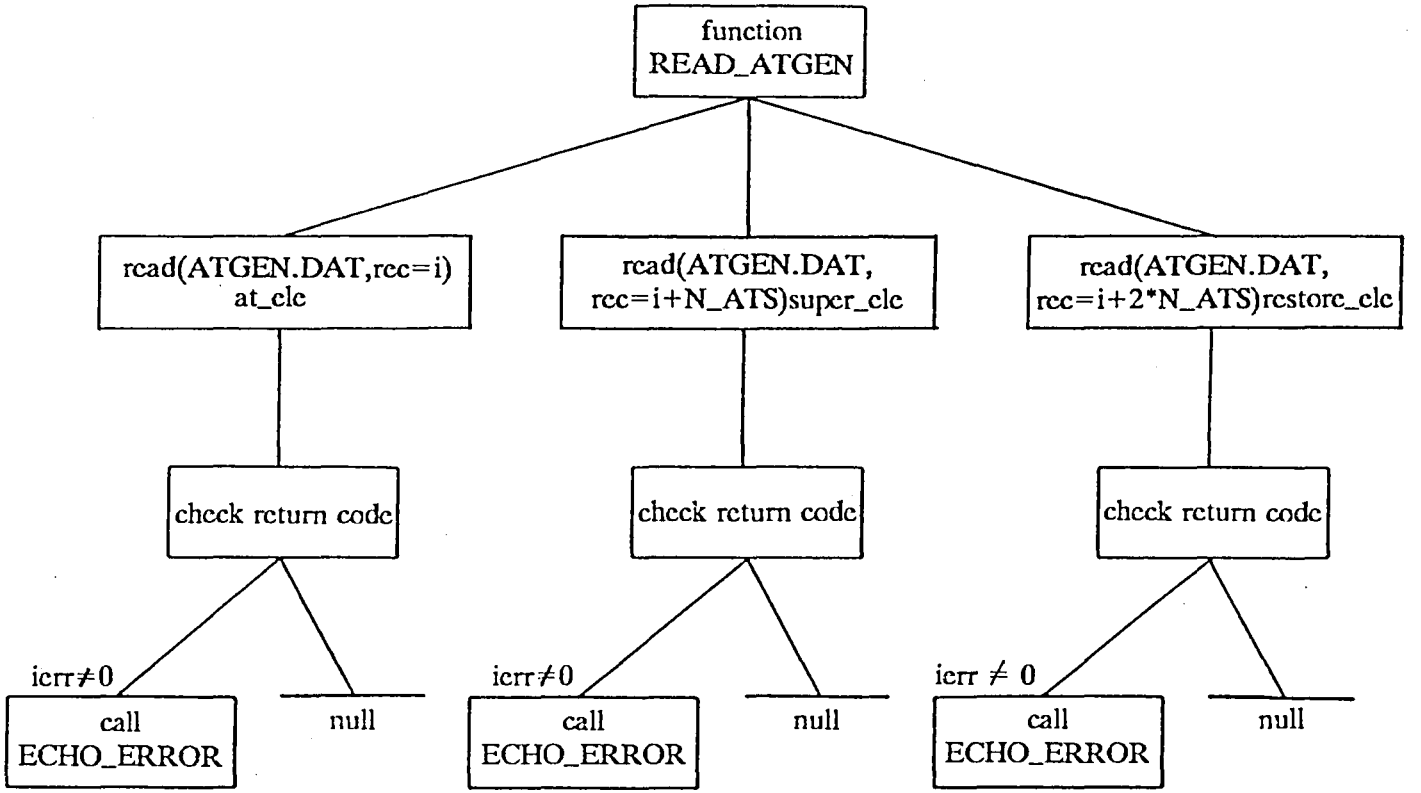
ECHO_ERROR



externals:

SET_MASK
FORLINREP.COM
ECHO_ERROR

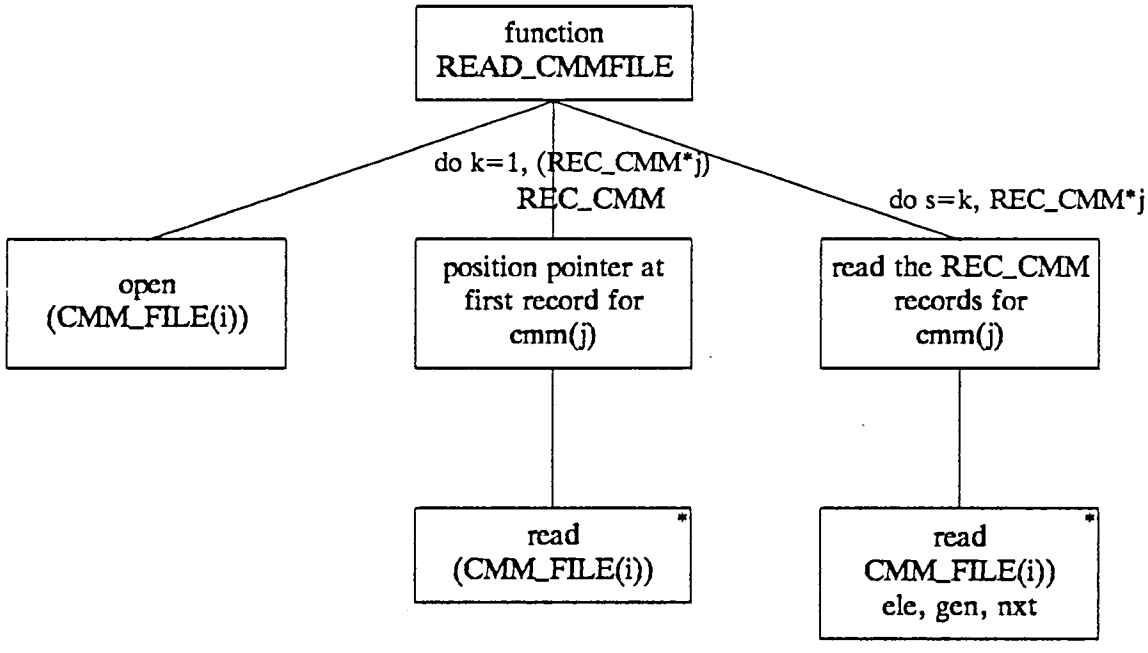
module: READ_ATGEN



Externals:

ECHO_ERROR

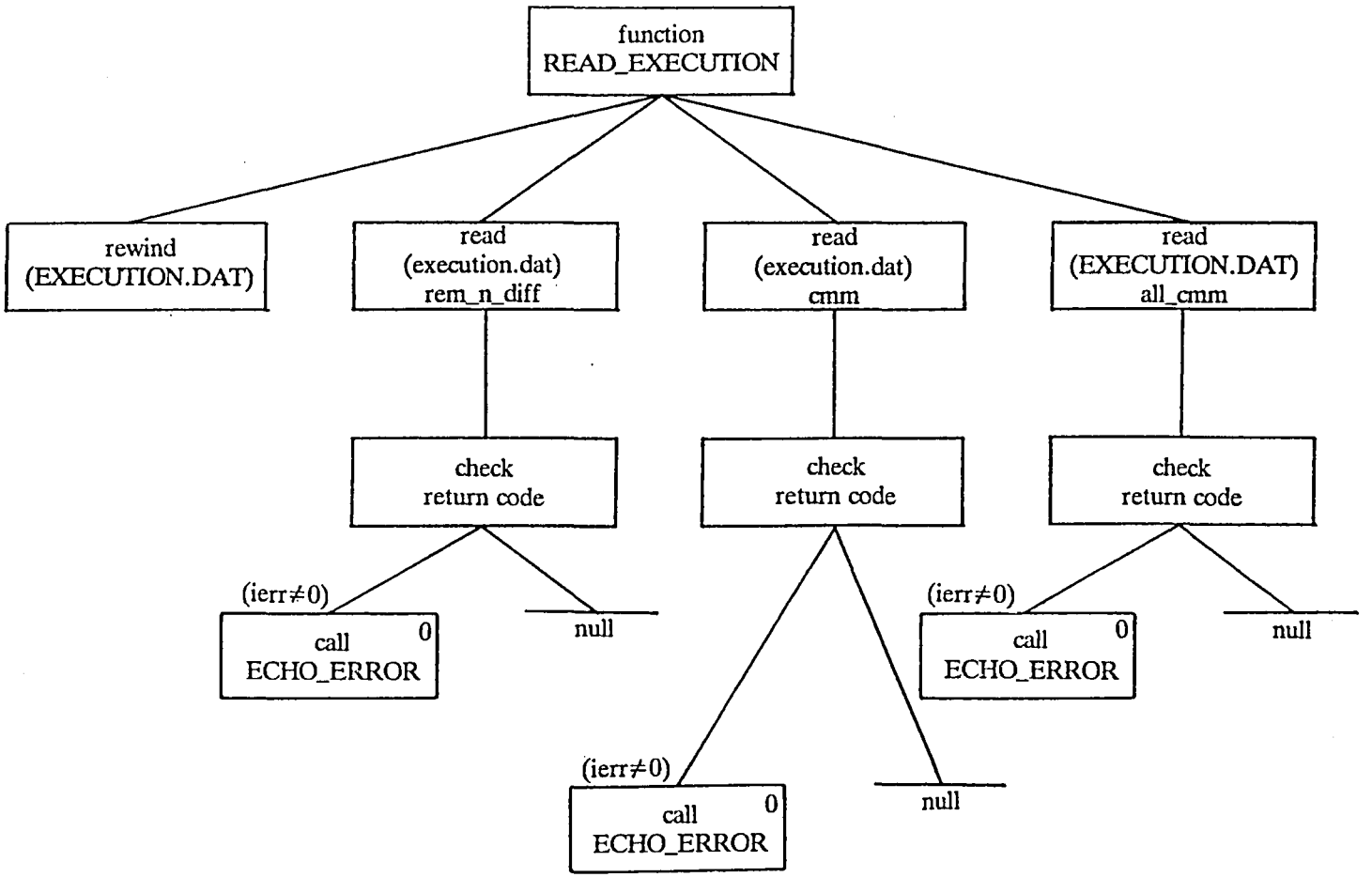
module: READ_CMMFILE



Externals:

none

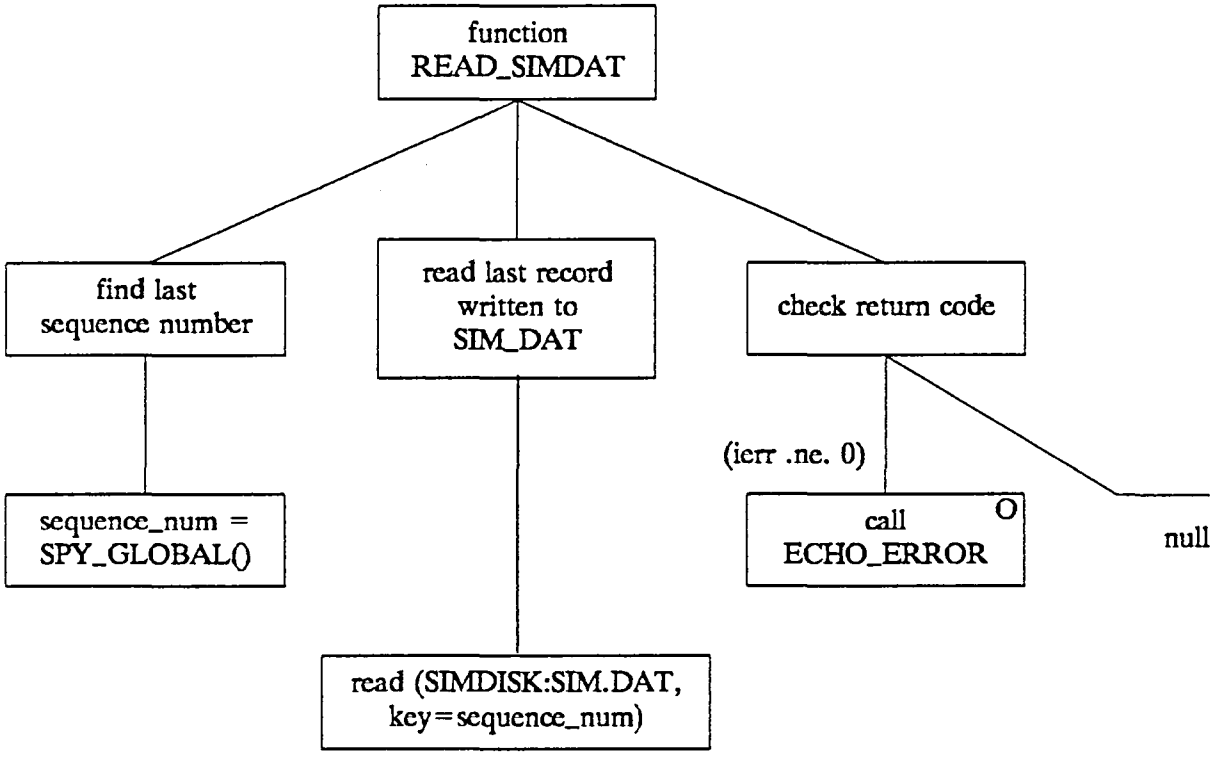
module: READ_EXECUTION



externals:

none

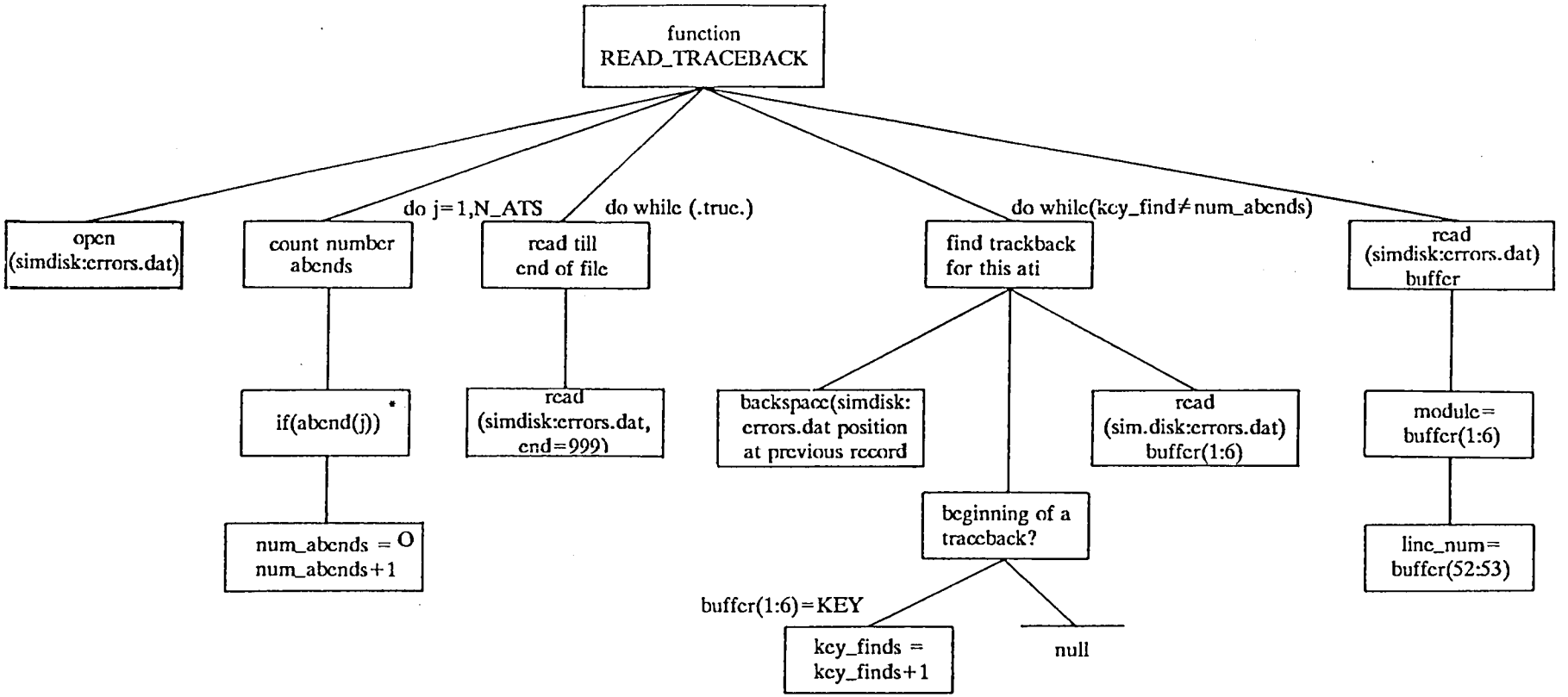
module: READ_SIMDATA



Externals:

SPY_GLOBAL

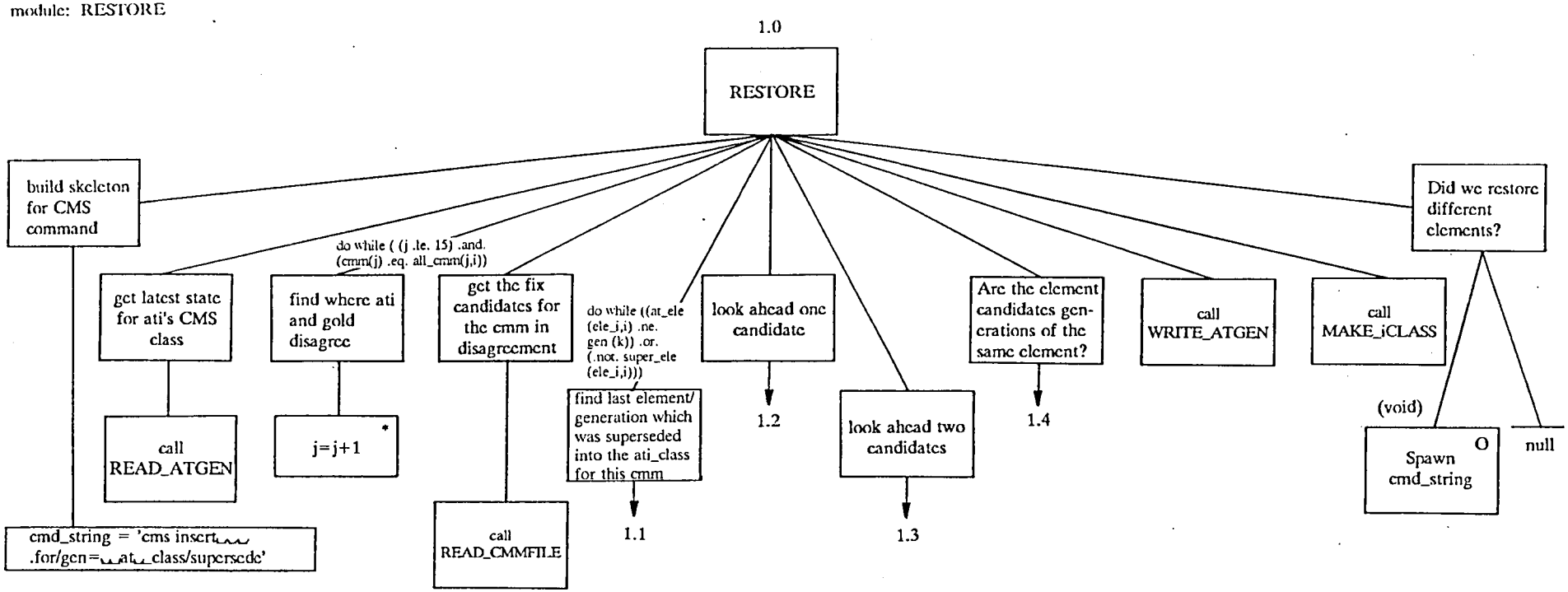
module READ_TRACEBACK



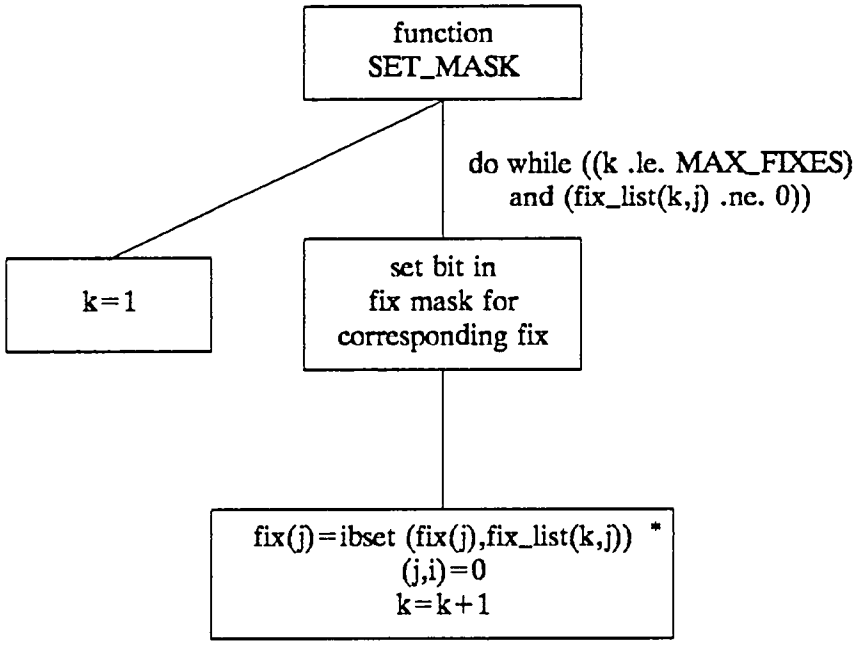
External:

none

module: RESTORE



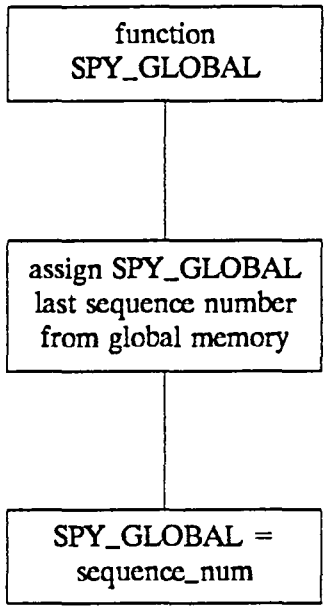
module: SET_MASK



Externals:

none

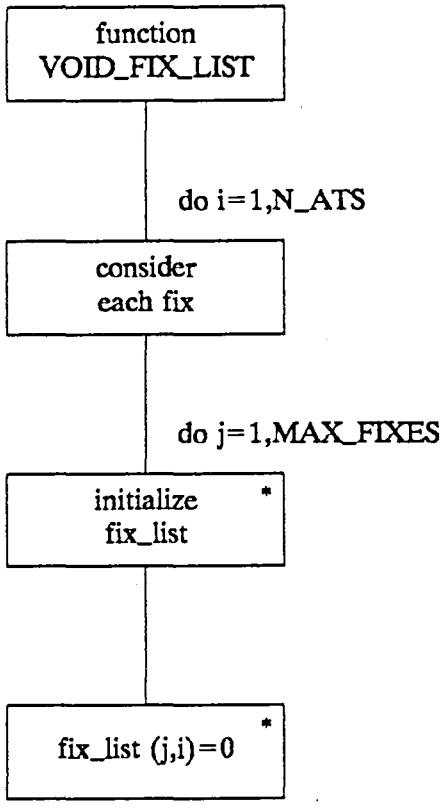
module: SPY_GLOBAL



Externals:

none

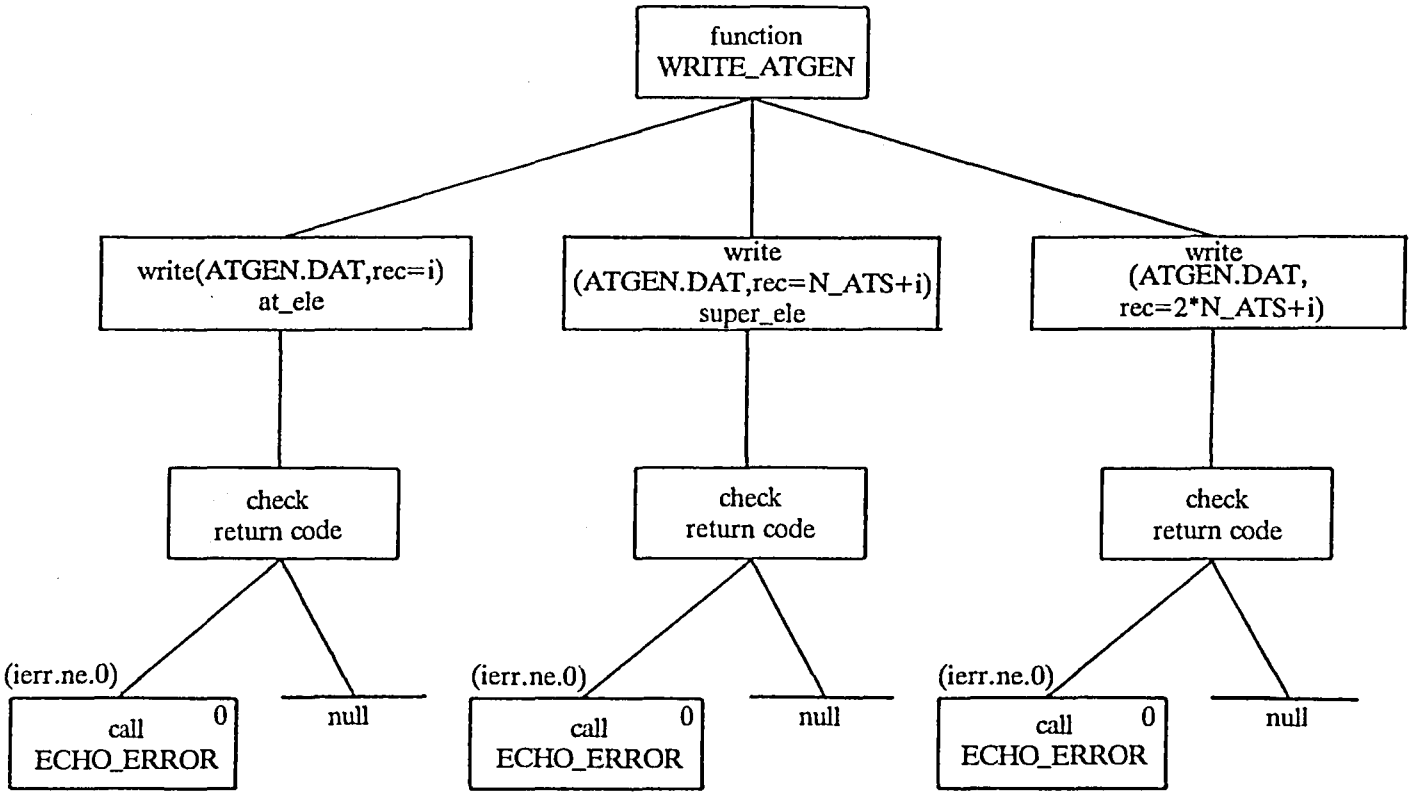
module: VOID_FIX_LIST



Externals:

none

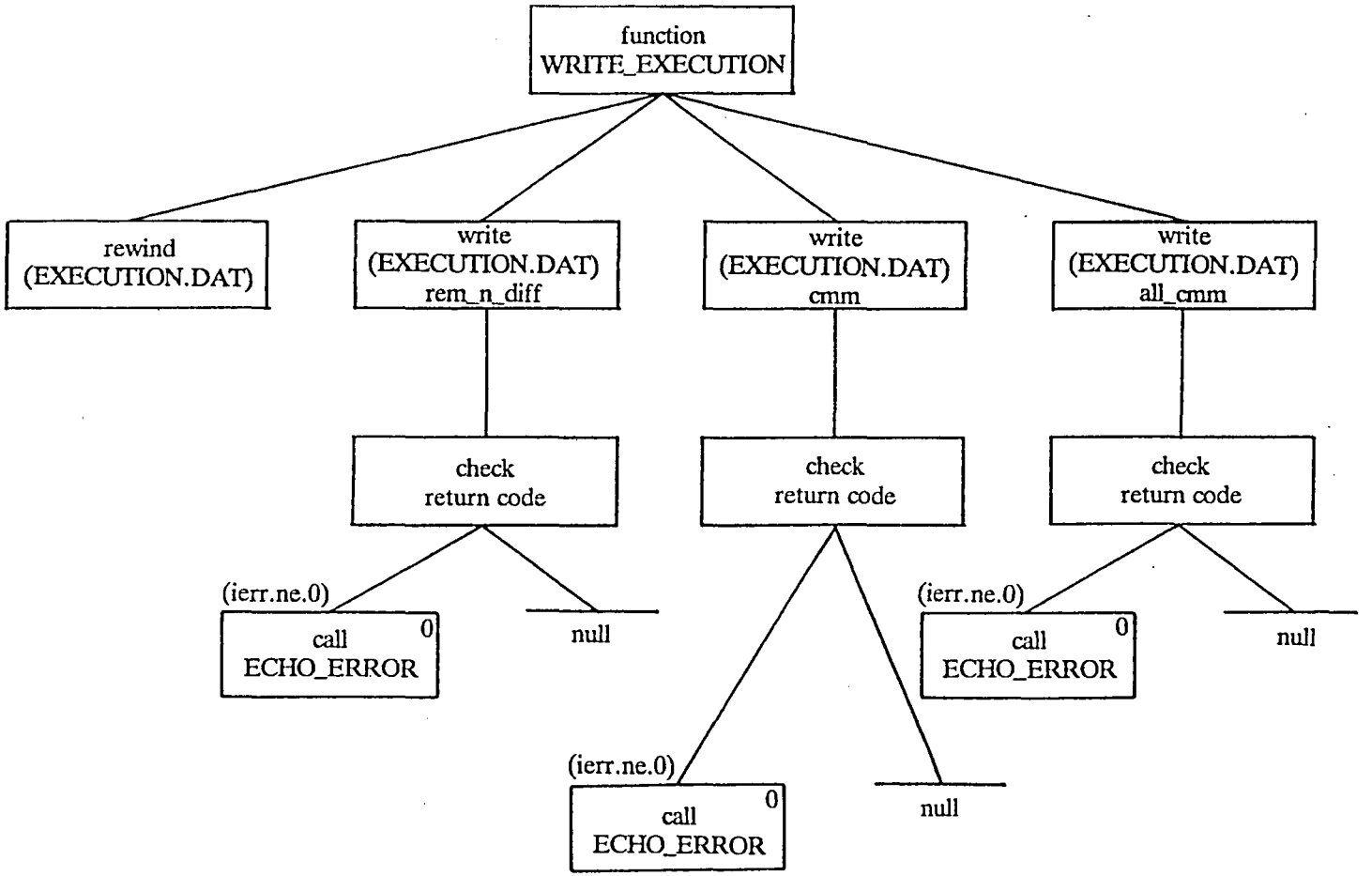
module: WRITE_ATGEN



externals:

ECHO_ERROR

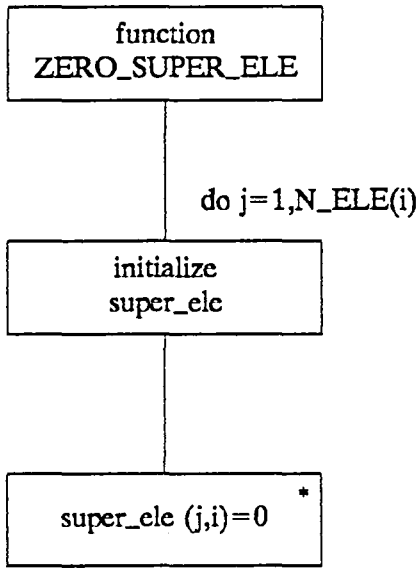
module: WRITE_EXECUTION



externals:

ECHO_ERROR

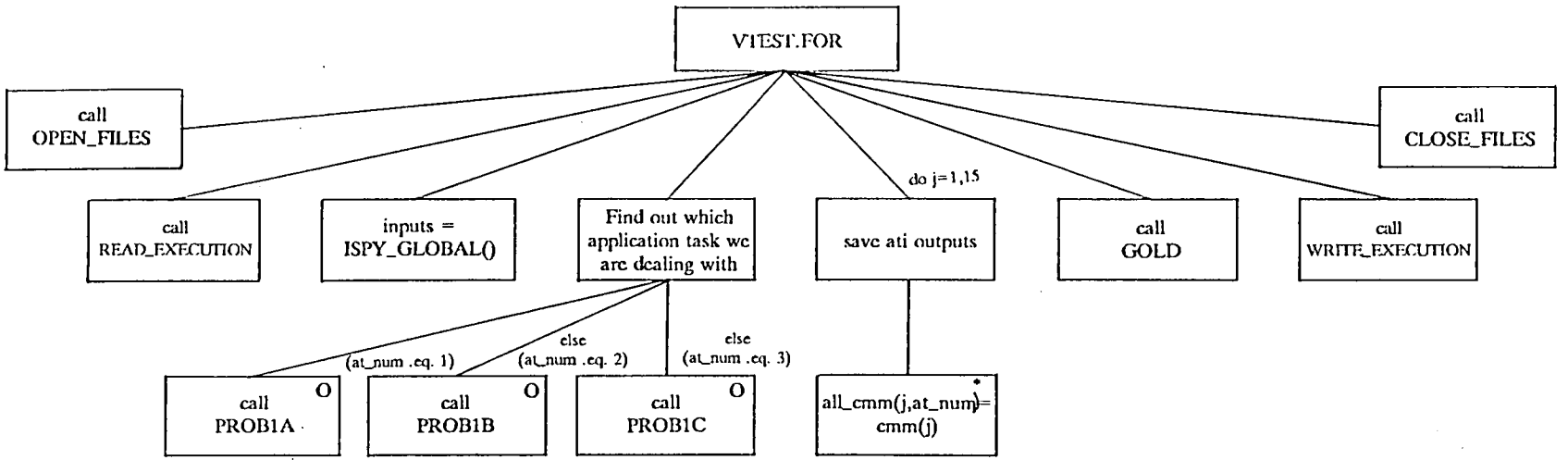
module: ZERO_SUPER_ELE



Externals:

none

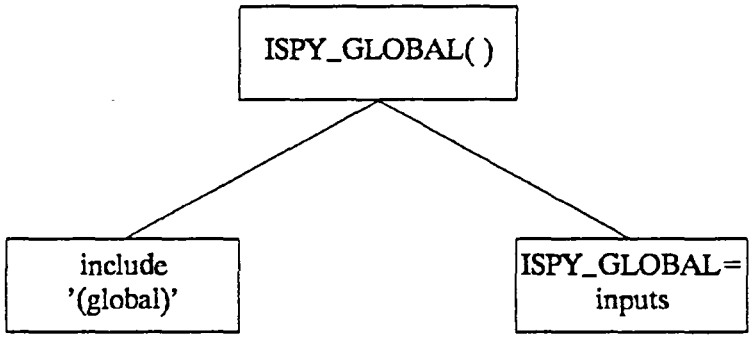
module: VTEST.FOR



*AUTOSIM SPECIFIC - reads from GLOBAL.DAT

Externals:

OPEN_FILES
READ_EXECUTION
ISPY_GLOBAL()
PROB1i
GOLD
WRITE_EXECUTION
CLOSE_FILES


ISPY_GLOBAL( )
include
'(global)'
ISPY_GLOBAL=
inputs

APPENDIX B: AUTOSIM FILE DESCRIPTIONS

ATGEN.DAT:

purpose:

Keep track of what CMS element generations are presently installed in the ati_class and what CMS element generations were previously installed in the ati_class.

background:

Each of application task has its own CMS class, i.e. there are N_ATS CMS classes defined within the CMS library ASIM_CMS_LIB. There are N_ELE(i) elements in the CMS class corresponding to ATi. N_ELE is an N_ATS integer*4 array defined in AUTO.INC along with other data variables. ATGEN.DAT is organized into three different groups of records; each group has N_ATS records. All application tasks have one record within group of N_ATS records. Specifically, ATi has its records located at record positions i, i+N_ATS, and i+(2*N_ATS). NOTE! To enable this random access of records, ATGEN.DAT was created as a DIRECT access file. To insure proper file integrity DO NOT EDIT ATGEN.DAT. Records within the file ATGEN.DAT may be modified manually with the Fortran program MODATGEN.

reference:

ATGEN.DAT is referenced with unit number 3 in the open, read, and write statements to this file. the open statement is as follows.

```
open( unit=3, file=ATGEN.DAT, access='DIRECT', status='OLD' )
```

data structures:

at_ele - a 17 by N_ATS element byte array containing the fisrt group of N_ATS records. This group of records indicates what generations of the CMS elements are currently installed in the CMS class for each application task.

super_ele - a 17 by N_ATS element byte array containing the second group of N_ATS records. This group of records indicates what CMS class elements are to be superseded with new generations.

restore_ele - a 17 by N_ATS element byte array containing the third group of N_ATS records. This group of records indicates what generations of the CMS elements were installed in the previous version of the CMS class for each application task.

record formats:

first group: at_ele -or-
at1's (10i3)
at2's (01i3)

```
                at3's      (17i3)
second group: super_ele   -or-
                at1's      (10i3)
                at2's      (01i3)
                at3's      (17i3)
third  group: restore_ele -or-
                at1's      (10i3)
                at2's      (01i3)
                at3's      (17i3)
```

AUTOERR.DAT:

purpose:
log AUTOSIM errors of the nature (1) out fixes for an ATi, or (2) open, read, or write errors.

reference:
AUTOERR.DAT is referenced with unit number 10 in the following open statement.

```
open(unit=10, file='AUTOERR.DAT', access='SEQUENTIAL', status='OLD')
```

data structures:
module - character*(*), the module in which the error occurred

file - character*(*), the file that was be referenced when the error occured

fail_msg - the message accompanying the failure (i.e. open, read,...)

iostat - i*4, either the return status code from a open, read,... , or the CMM that did not have any more fixes

stamp - character*23, the time the failure occurred

record format:

```
stamp
-or- (a)        !time in format  dd-mmm-yyyy hh:mm:ss.cc
module  file  fail_msg  iostat
-or- ("ERROR: <', a, '> ', a, 1x, a, '; iostat = ', i10)
```

EXECUTION.DAT:

purpose:
serves as the communication link between the spawned subprocess VTEST and the AUTOSIM program. VTEST updates the last two records which contain the output from the GOLD and the ATi's CMMs. MEAS_IMP, a function of AUTOSIM, updates the first record which contains the number of differences between the GOLD's and the ATi's CMMs.

reference:
EXECUTION.DAT is referenced as unit 4 in the following statement:

```
open(unit=4, file='EXECUTION.DAT', access='SEQUENTIAL', status='OLD')
```

data structures:
rem_n_diff - a three element integer*4 array containing the number of differences between the GOLD's CMMs and a particular ATi's. the valid range of values for these fields is a integer greater than or equal to 0 but less than or equal to 15.

cmm - a fifteen element integer*4 array containing the GOLD's CMM outputs. each element in this array has the value of either a 0 or a 1.

all_cmm - an N_ATS by fifteen element integer*4 array containing each of the ATis' CMM outputs. each element in this array has the value of either a 0 or a 1.

record formats:
first record: rem_n_diff (' ', 3i3)
second record: cmm (' ', 15(i1,','))
third record: all_cmm (' ', 45(i1,','))

iABEND.DAT:

purpose:
The iABEND.DAT files contain a history of the abend failures which have been documented for each ATi during the execution the Launch Interceptor Code.

reference:
the iABEND.DAT files are referenced individually with unit number 30 in the following type of open statement:

```
open( unit=20, file=iABEND.DAT, access='SEQUENTIAL', status='OLD' )
```

data structures:
function - a character string of length six. this variable contains the the name of the module in which the abend occures.

position - a character string of length two. this variable contains the where in the module the abend occurred.

ele(1) - a character string of length three. this variable contains the CMS element with the source for the particular module under consideration.

gen(1) - a byte variable. the generation of the CMS element with the fix required to fix the abend in the given module.

record format:
function position ele(1) gen(1)
-or- (t1,a6,t9,a2,t13,a3,t18,i2)

iCLASS.DAT:

purpose:
the records of iCLASS.DAT indicate to the spawnned FIXAPP.COM which CMS elements are to be superseded with new generations, i.e. which fix to install.

reference:
the iCLASS.DAT files are referenced through the N_ATS integer*4 data array AT_UNIT. each ATi's iCLASS.DAT file is assigned a unit number which is stored in the ith element of the AT_UNIT array. currently, AT_UNIT(1)=11, AT_UNIT(2)=12, AT_UNIT(3)=13. the iCLASS.DAT files are made available through the statement:

```
      open( unit=AT_UNIT(i), file=iCLASS.DAT, access='SEQUENTIAL',
     +      status='OLD' )
```

data structures:
ele_gen - character*10, the CMS element name and generation which is to be or was superseded in the CMS class for the ATi presently under consideration.

record format:
ele_gen -or- (1x, 1a1, 1i2.2, '.for(', 1i1, ')')

iCLOBBER.DAT:

purpose:
The iCLOBBER.DAT files contain a history of the overwrite failures which have been doucmented for each ATi during the execution of the Launch Interceptor Code.

reference:
iCLOBBER.DAT files are referenced individually with unit number 20 in the following type of open statement:

```
open( unit=20, file=iCLOBBER.DAT, access='SEQUENTIAL', status='OLD' )
```

data structures:
clob - integer*4, the clobber value indicating which common region was overwritten

ele(1) - character*3, the CMS element containing this section of code with the common region in which the overwrite occurred

gen(1) - byte, the generation of CMS element from above with the fix to stop the overwrite from occurring again

record format:
clob ele(1) gen(1)
- or - (t1,i1,t4,a3,t9,i1)

iCMM.DAT:

purpose:
The iCMM.DAT files contain a history of failures and associated fixes in the conditions met matrix which have been recorded for each ATi during the execution of the Launch Interceptor Code.

reference:
iCMM.DAT files are referenced individually with unit number 40 in the following type of open statement:

```
      open( unit=40, file=iCMM.DAT, access='SEQUENTIAL',
     +      status='OLD', recl=24 )
```

data structures:
ele(REC_CMM) - a REC_CMM element character*3 array, the CMS elements which contain the section of code in which it is possible for this cmm to fail

gen(REC_CMM) - a REC_CMM byte array, the generation of the CMS elements from above with potential fix(es) for the failed cmm

nxt(REC_CMM) - a REC_CMM byte array; when searching for a fix, we find the element/generation which is currently installed, and index off this 'nxt' field of the same record to find the next potential fix, provided another fix exists.

record formats:
ele gen nxt
-or- (t4, a3, t9, i2, t13, i2)

iLAUNCH.DAT:

purpose:
iLAUNCH.DAT files are used when an ATi fails and the failure does not show up as an abend, overwrite, or as disagreement in the conditions met matrix; a launch error is detected by the assertion of a .not.constraints_met element and the index to the all_cmm array being out of bounds.

reference:
iLAUNCH.DAT files are referenced individually with unit number 50 in the following type of open statement

```
open(unit=50, file=iLAUNCH.DAT, access='SEQUENTIAL', status='OLD')
```

assumptions:
only one launch error for any ATi

data structures:
ele(1) - a character string of length three. this variable contains the CMS element with the source code where the launch error occures

gen(1) - a byte variable. the generation of the CMS element with the fix required to correct the launch error in the given module.

record format:
ele(1) gen(1)
-or- (t3, a3, t8, i2)

SIM.DAT:

purpose:

records a history of simulator failures by storing the environmental parameters, generated inputs, and simulator and application tasks' outputs. This makes it possible for an operator to determine why the simulator failed by examining design stages of previous replications.

reference:

SIM.DAT is referenced as unit 1 by all modules in the Autosim, Interface, and the Simulator. SIM.DAT is an indexed file; so, unit 1 is accessed with the use of keys. The primary key is the sequence number, i.e. the sequential number of a record in the file. Modules referencing SIM.DAT have read and write privledges to the file. The file is expected to exist in the SIMDISK: directory, and an error message will result if the file is not found.

data structures:

the common regions of SIM.DAT:

inputs - the generated inputs consist of the following variables:

```
x,y,el,r,eps2,a,m,q,eps1,n1,n2,m2,n3,m3,n4,
m4,n6,bigl,bigr,bige,bign,lcm,pumdia,p,ifout
```

outputs - the simulator outputs consist of the following variables:

```
cmm,fum,launch,pum
```

allouts - the application task outpts consist of the variables:

```
all_cmm,all_fum,all_launch,all_pum
```

all_voterouts - ouput from the voters

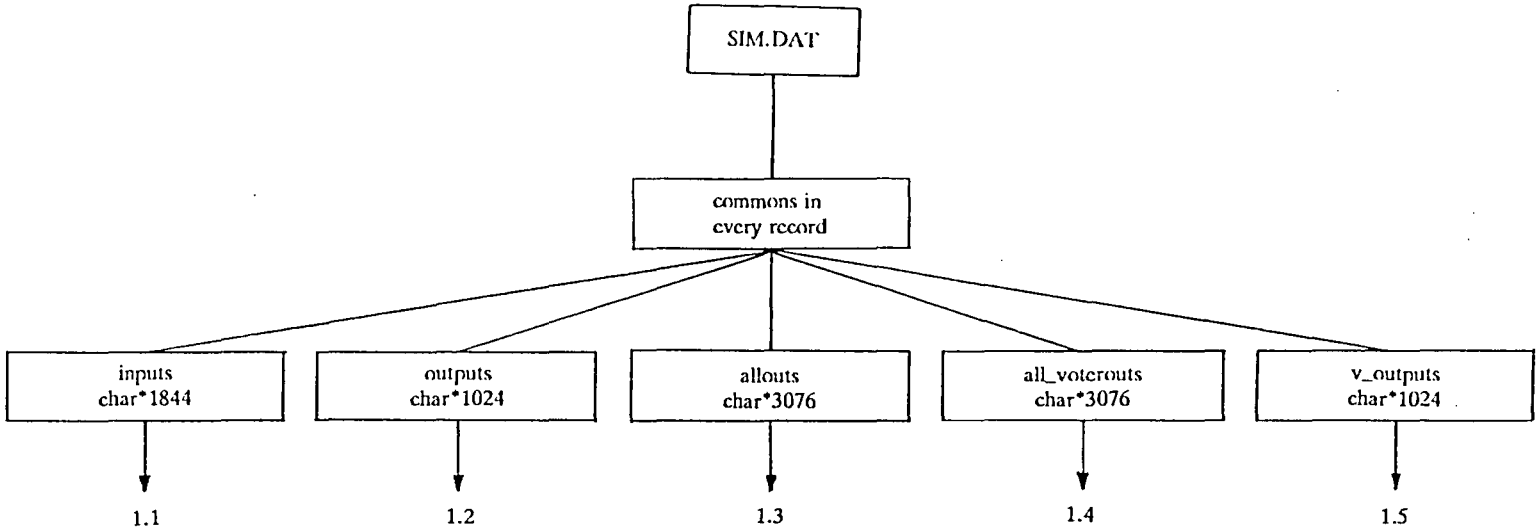
```
all_v_cmm,all_v_fum,all_v_launch,all_v_pum,
all_v_comp_launch
```

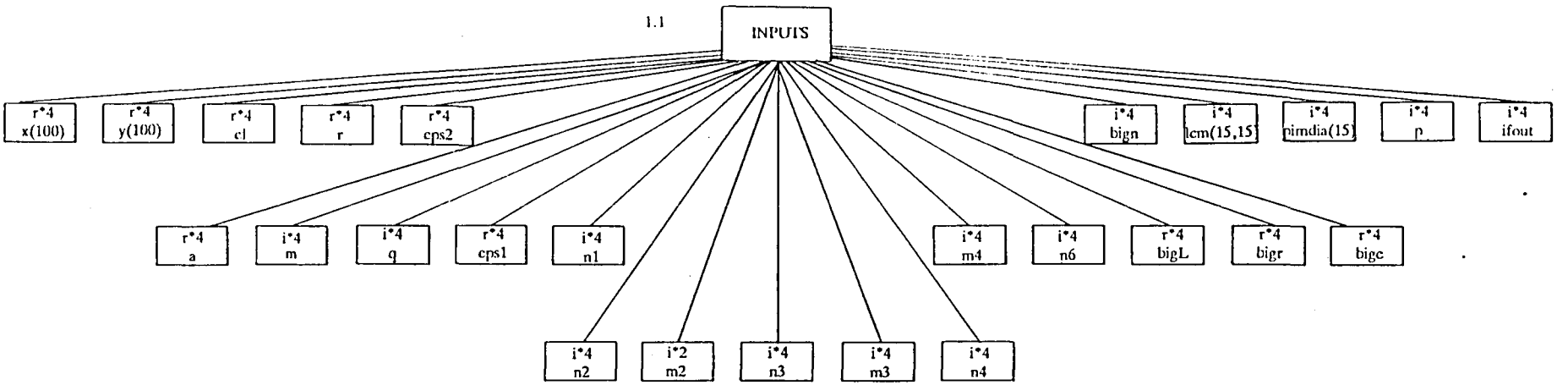
record format:

the simulator writes a SIM.DAT record with the following statement, the format is implied.

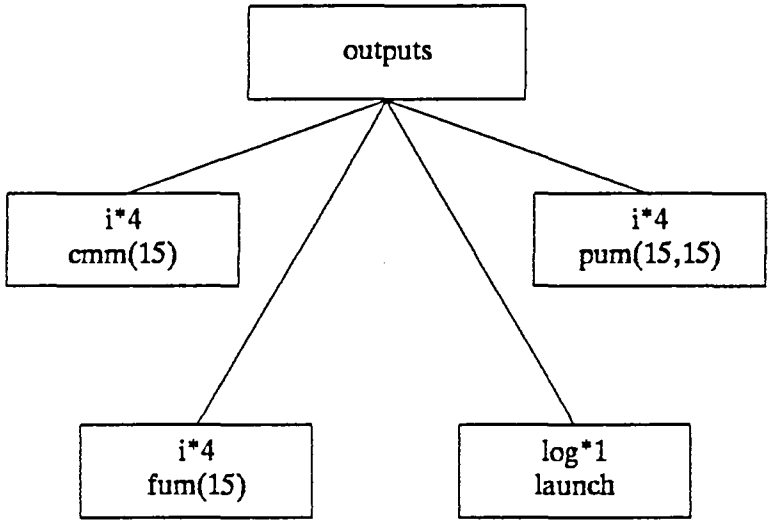
```
write(unit=1,iostat=iret)list,inputs,outputs,allouts,all
```

1.0

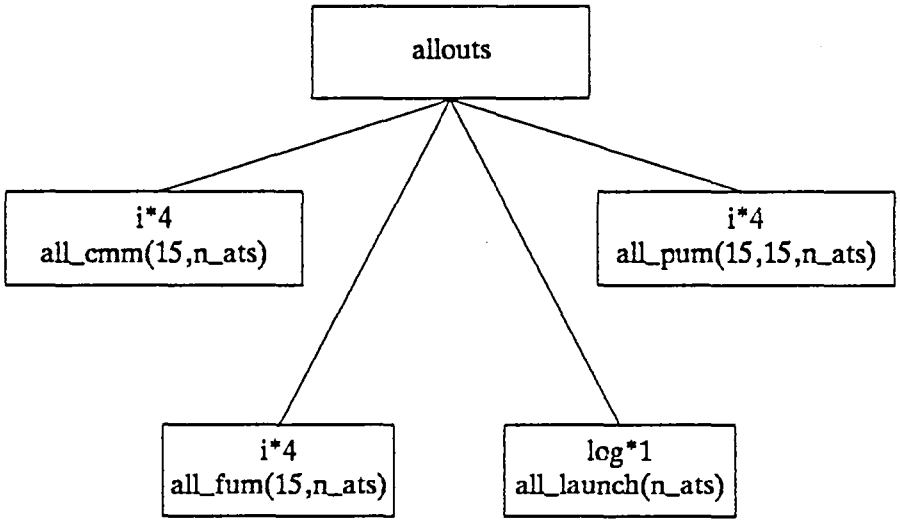

SIM.DAT
commons in
every record
inputs
char*1844
outputs
char*1024
allouts
char*3076
all_voterouts
char*3076
v_outputs
char*1024
1.1
1.2
1.3
1.4
1.5

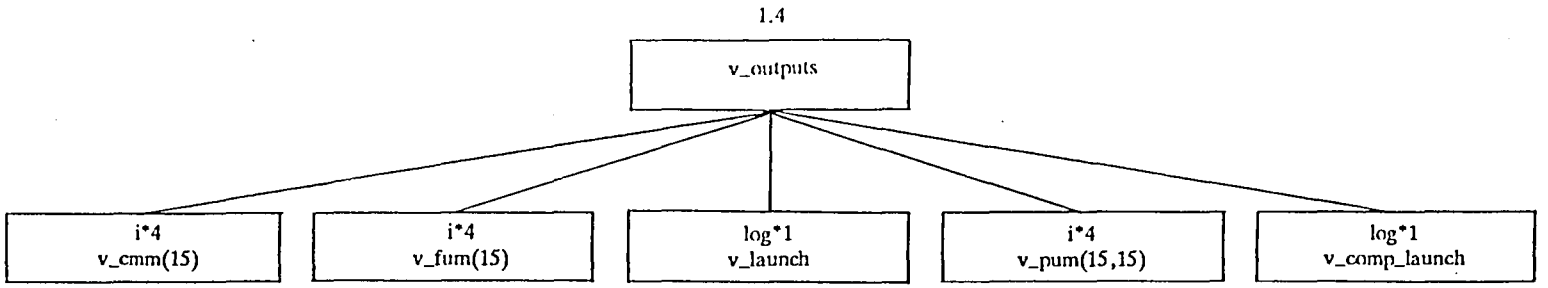

1.1
INPUTS
r*4 x(100)
r*4 y(100)
r*4 cl
r*4 r
r*4 eps2
i*4 bign
i*4 lcm(15,15)
i*4 pimdia(15)
i*4 p
i*4 ifout
r*4 a
i*4 m
i*4 q
r*4 eps1
i*4 n1
i*4 m4
i*4 n6
r*4 bigL
r*4 bigr
r*4 bige
i*4 n2
i*2 m2
i*4 n3
i*4 m3
i*4 n4

1.2

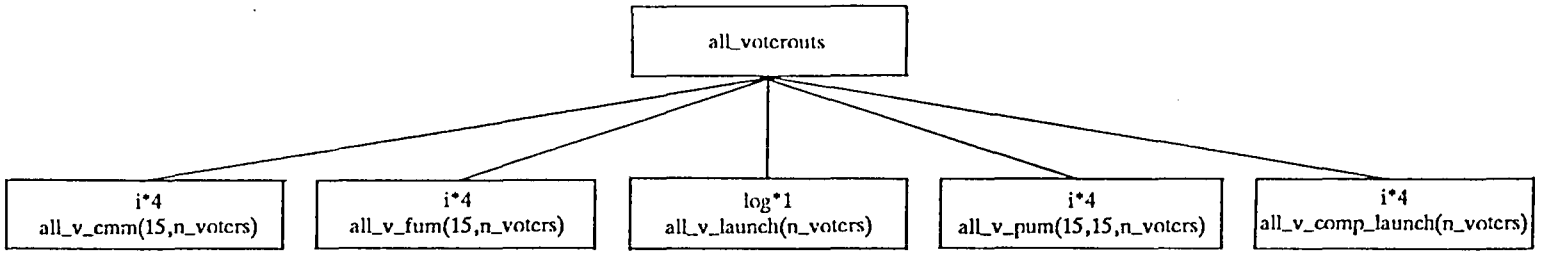


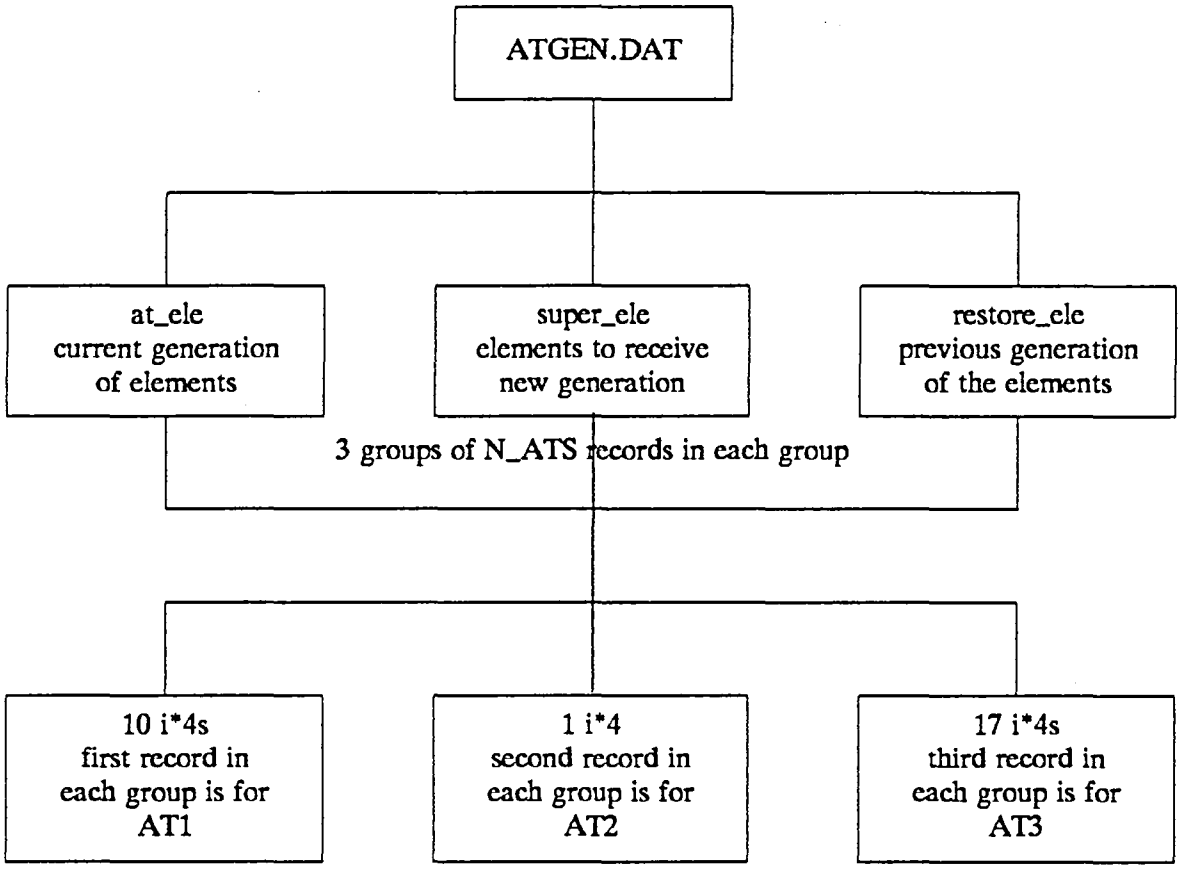
1.3


allouts
i*4
all_cmm(15,n_ats)
i*4
all_pum(15,15,n_ats)
i*4
all_fum(15,n_ats)
log*1
all_launch(n_ats)

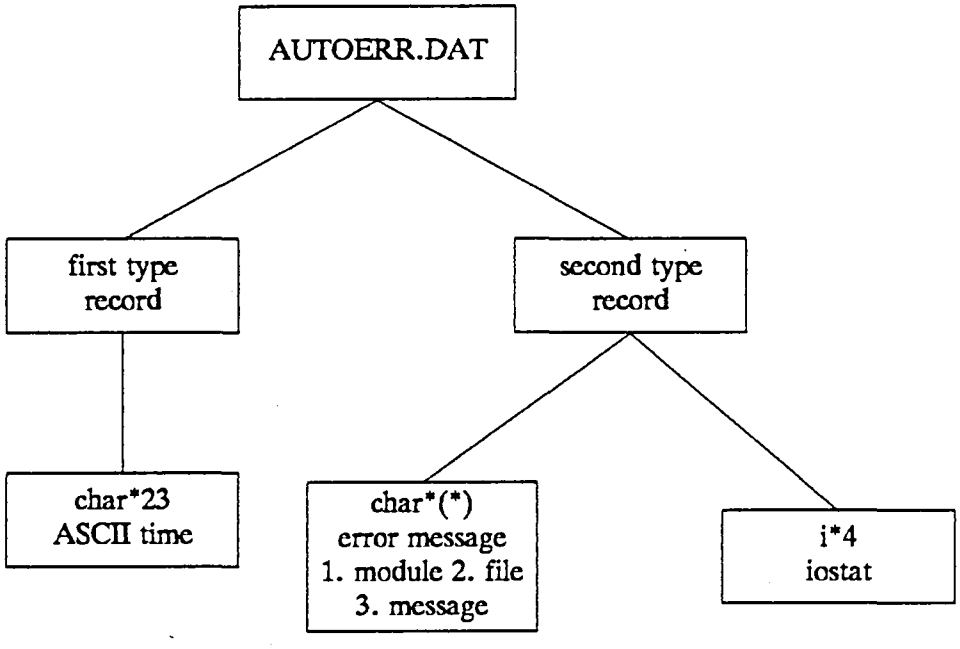

1.4
v_outputs
i*4
v_cmm(15)
i*4
v_fum(15)
log*1
v_launch
i*4
v_pum(15,15)
log*1
v_comp_launch

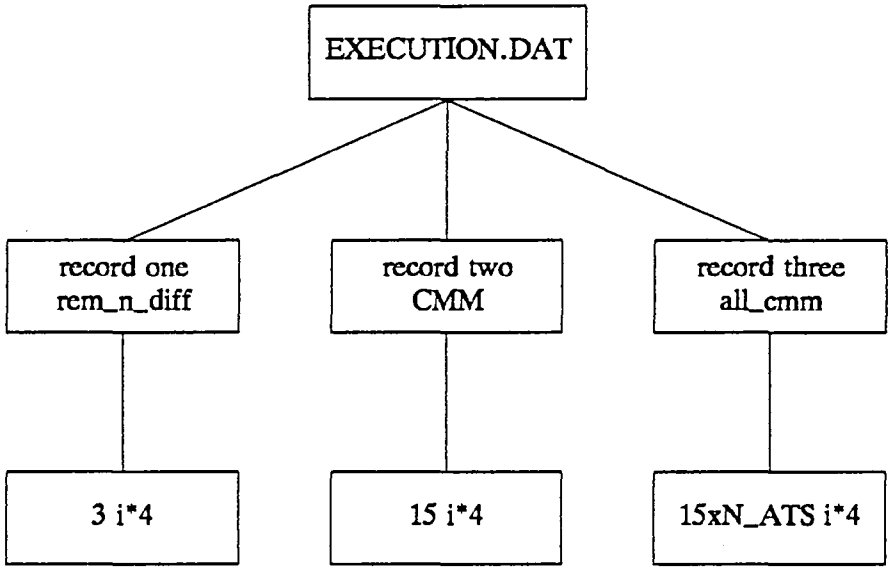
1.5

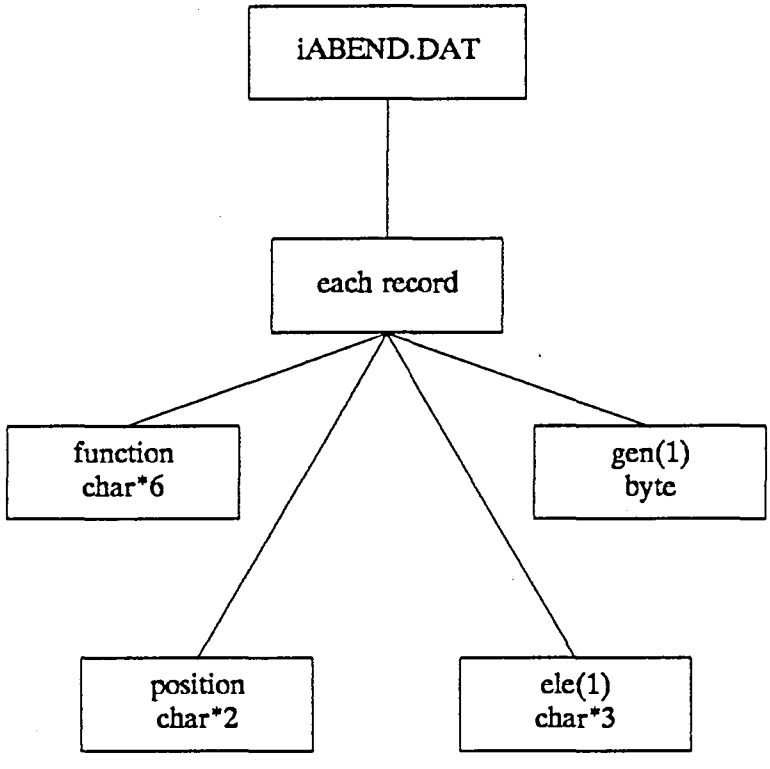

all_voterouts
i*4
all_v_cmm(15,n_voters)
i*4
all_v_fum(15,n_voters)
log*1
all_v_launch(n_voters)
i*4
all_v_pum(15,15,n_voters)
i*4
all_v_comp_launch(n_voters)

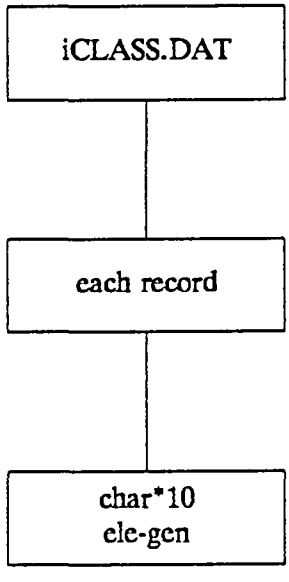


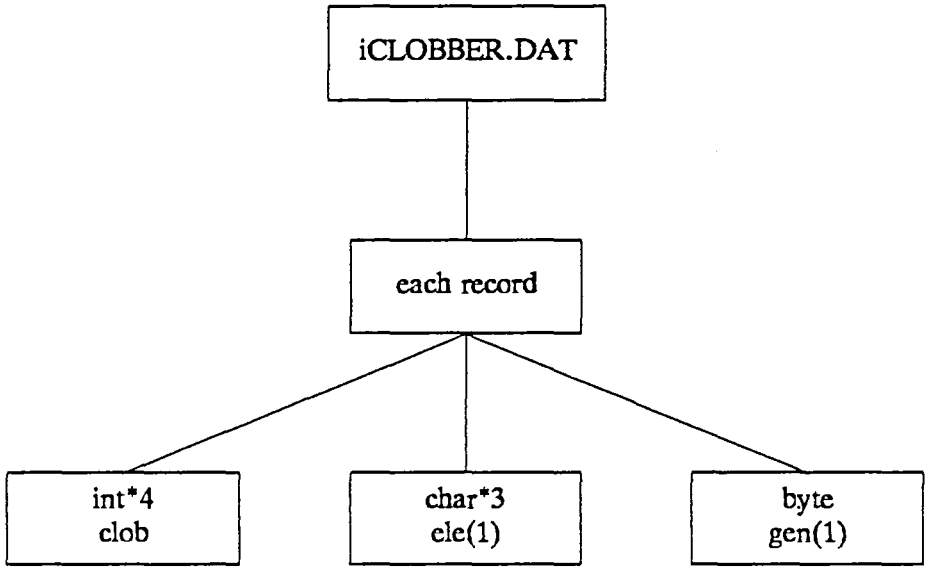
* the three groups of N_ATS records is independent of the N_ATS

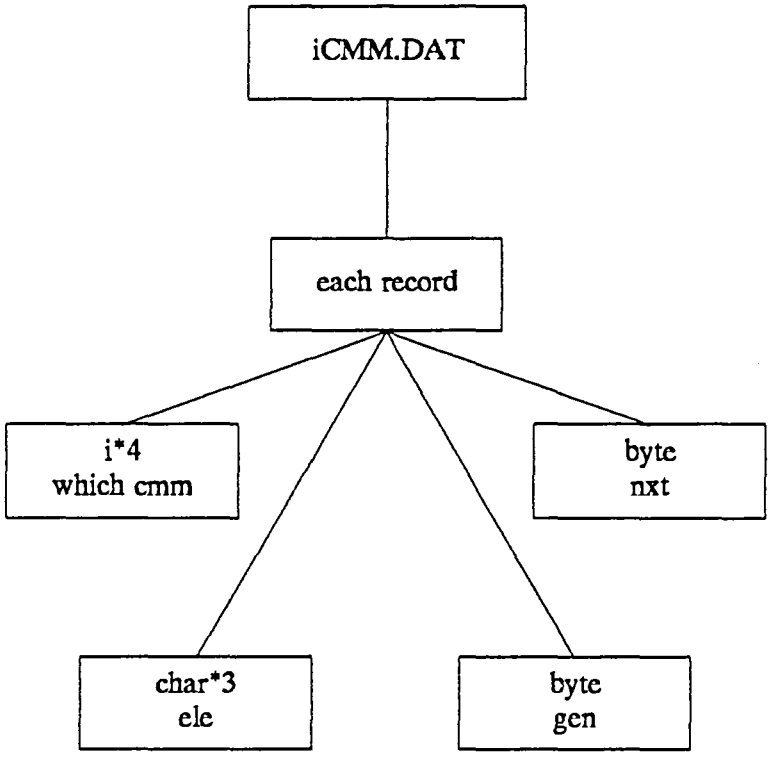

AUTOERR.DAT
first type record
second type record
char*23 ASCII time
char*(*) error message 1. module 2. file 3. message
i*4 iostat

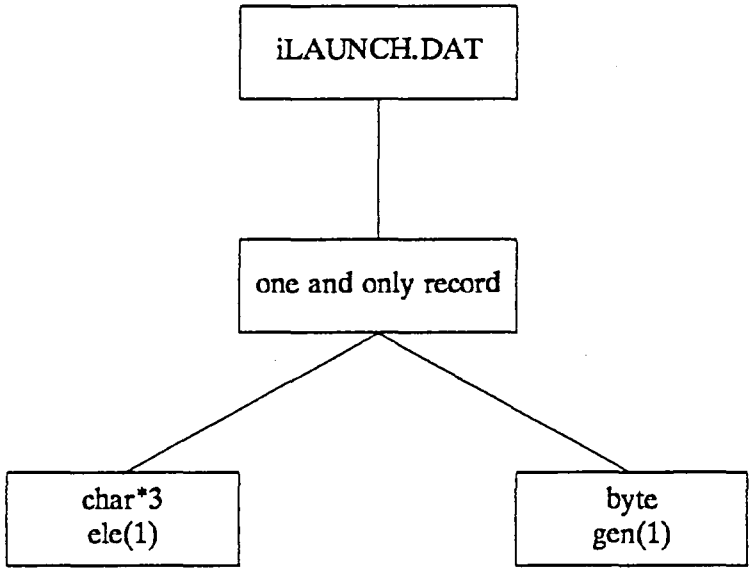

EXECUTION.DAT
record one
rem_n_diff
record two
CMM
record three
all_cmm
3 i*4
15 i*4
15xN_ATS i*4

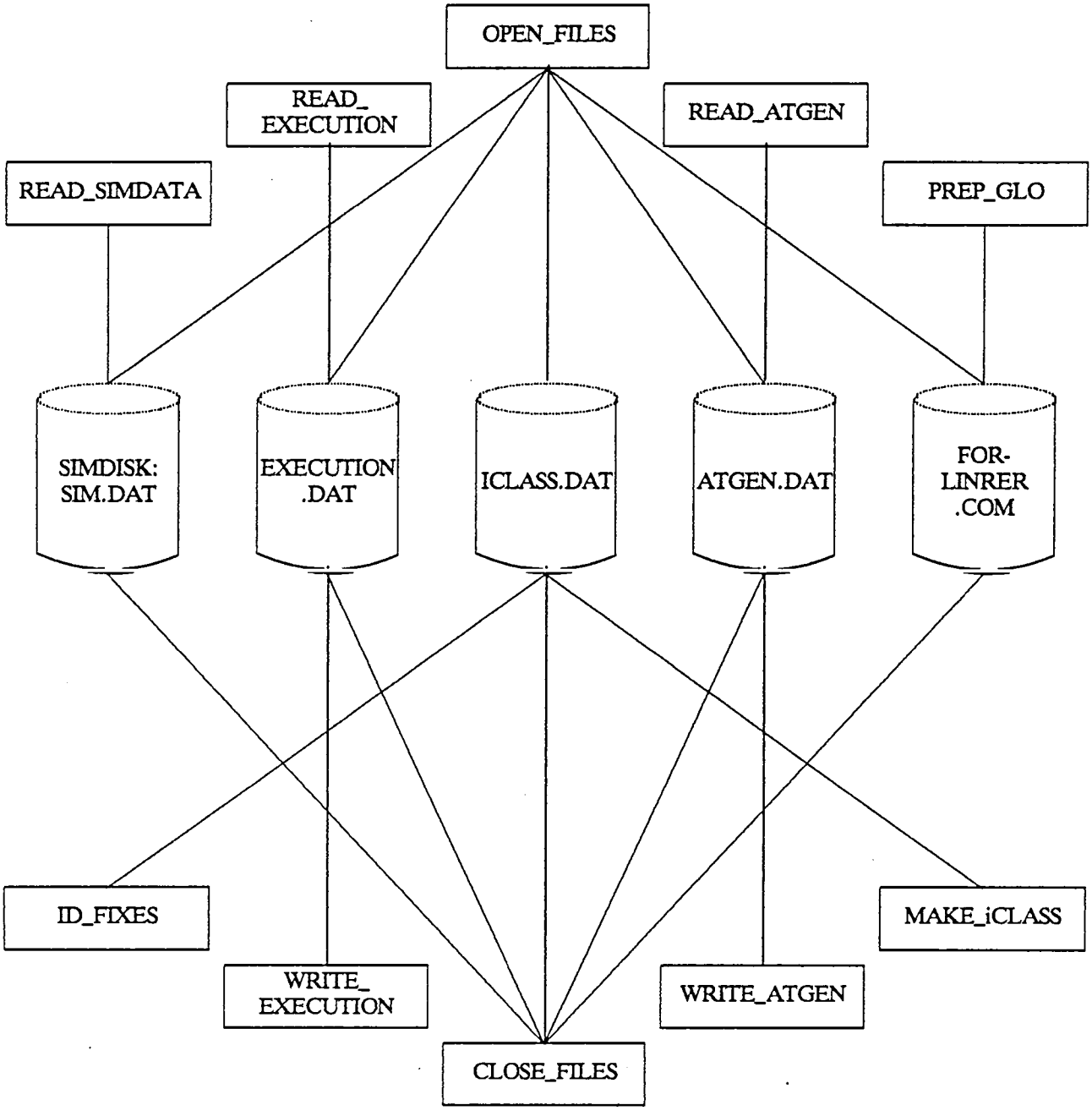

iABEND.DAT
each record
function
char*6
gen(1)
byte
position
char*2
ele(1)
char*3


iCLASS.DAT
each record
char*10
ele-gen

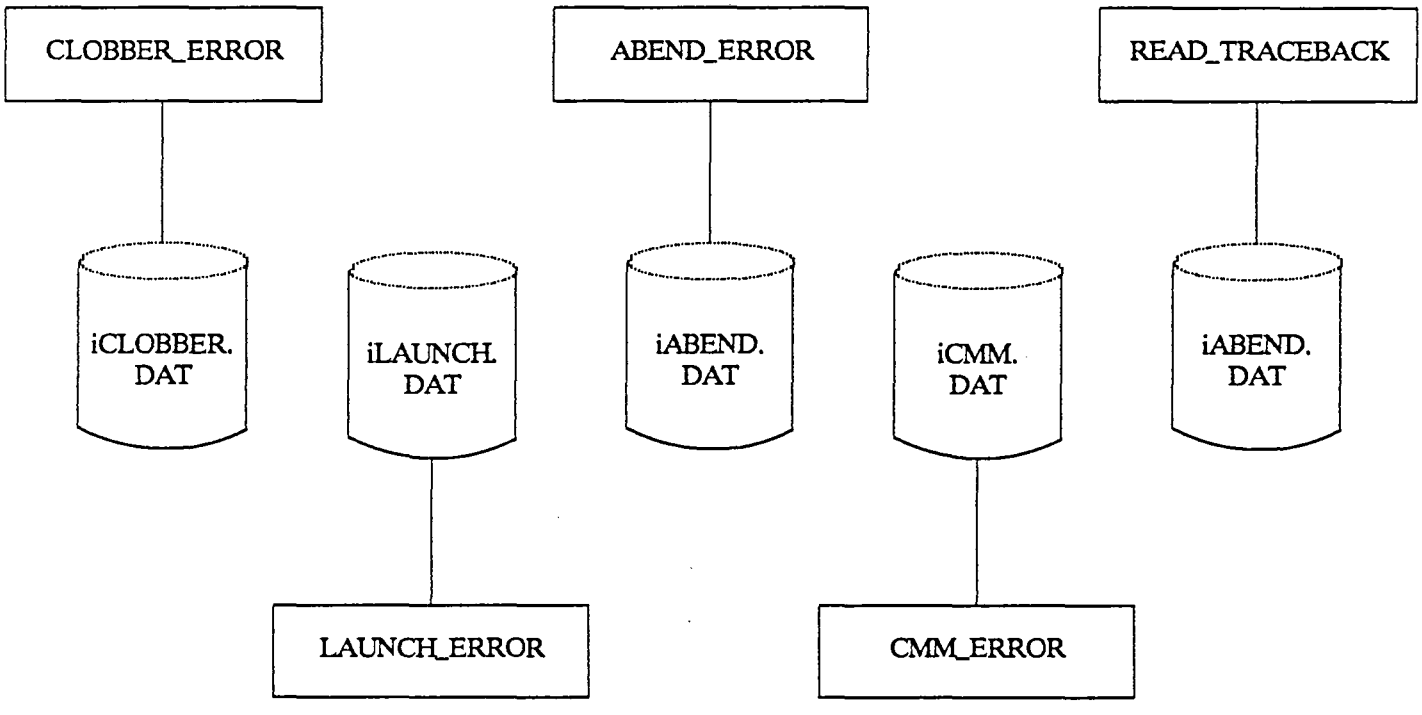

iCLOBBER.DAT
each record
int*4
clob
char*3
ele(1)
byte
gen(1)


iCMM.DAT
each record
i*4
which cmm
byte
nxt
char*3
ele
byte
gen


iLAUNCH.DAT
one and only record
char*3
ele(1)
byte
gen(1)



Files referenced by more than one routine.



Files referenced by only one routine.

APPENDIX C: LISTING OF AUTOSIM COMMAND PROCEDURES

AUTOSIM.COM

```
$ assign nla0: sys$input
$ assign out.dat sys$output
$ set default sim_auto_1
$ set rms/extend=3
$ cms set library asim_cms_lib
$ @[sim.prob1.auto.tools]bang
$ set process/name=autosim
$ run sim_auto_1:autosim
$ exit
```

FORLINREP.COM

```
$!!!!!!!!!
$ dtim = f$time()
$open/append chart chart.fil
$ write chart dtim
$ write chart "forlinrep.com"
$ close chart
$!!!!!!!!!!
$ delete vtest.map;*,vtest.lis;*
$ set rms/extend=3
$ fortran/list/continuations=99 prob1a
$ lib/rep simdisk:prob1lib.olb prob1a
$ fortran/list/continuations=99 prob1b
$ lib/rep simdisk:prob1lib.olb prob1b
$ fortran/list/continuations=99 prob1c
$ lib/rep simdisk:prob1lib.olb prob1c
$ lin/map simdisk:sim,prob1lib/l,opt/opt -
  /exe=sim_auto_1:sim.exe
$ delete prob1*.obj;*
$ purge/keep=3 sim.exe,prob1*.lis
$ exit
```

SIMBATCH.COM

```
$!!!!!!!!
$ dtim = f$time()
$ open/append chart chart.fil
$ write chart dtim
$ write chart "simbatch.com"
$ write chart " "
$ close chart
$!
$ open/append log log.dat
$ write log "start simualtor: ", dtim
$ write log " "
$ close log
$!!!!!!!!!!!
$ set rms/extend=3
$ delete sim_auto_1:out.dat;*,sim.map;,simbatch.log;,for0*.dat;
$ delete asim_data:*class.dat;*
$ purge simdisk:inputs.dat
$ set process/priority=5
$ assign sim_auto_1:simbatch.log for006
$ set process /name=sim
$ run sim_auto_1:sim
$ exit
```

VTEST.COM

```
$!vtest.com - procedure to test a single version of an AT
$!!!!!!!!!!!!!!!!!!!!!!!!
$ open/append log log.dat
$ open/append chart chart.fil
$ write log "vtest"
$ dtim = f$time()
$ write chart dtim
$ write chart "vtest"
$ close chart
$ close log
$!!!!!!!!!!!!!!!!!!!!!!!!
$ set nover
$ set rms/extend=3
$ s1 = "a"
$ s2 = "b"
$ s3 = "c"
$ at = s'p1'
$ set ver
$ for/list/cont=99/check=all/obj=tmp prob1'at'.for;
$ lin/map vtest.sub,tmp,simdisk:prob1lib.olb/l,opt/opt,sim_auto_1:auto.olb/l
$ set nover
$ dele tmp.obj;*
$ open/write tmp_dat: tmp.dat
$ write tmp_dat: p1
$ close tmp_dat:
$ assign/user tmp.dat for$accept
$ run vtest
$ dele tmp.dat;*
$ dele vtest.exe;*
$ exit
$
```

APPENDIX D. LOG OF AUTOSIM VALIDATION TESTS

		-AT1-		-AT2-		-AT3-			
Rep. No.	Seq. No.	D.S.	Fix Nos.	D.S.	Fix Nos.	D.S.	Fix Nos.	CASE	ERROR
1	113	0	0	0	0	0	0	0	AT1: CMM (5) AT2: launch error AT3: CMM (7)
	114	1	1,2	1	1	1	1,16	4	AT3: CMM (5,9,10,11, 12,13,14,15)
	115					2	2,6,9 10,11,12, 14,15	42	AT1: CMM (7)
	116	2	3					75	ABEND AT3 RADCTR line 58
	117					3	18	90	AT3: (4,8,13)
	118					4	5,8,13	100	AT1: CMM (8,13)
	119	3	6					150	ABEND AT1 RAD line 30
	120	4	5					203	AT1: CMM (10)
	121	5	7					1475	AT1: CMM (5)
	122	6						2641	AT3 ABEND
	123					5	17	2985	ABEND AT1
	124	7	8					10,000	End of Rep.
2	125	0	0	0	0	0	0	0	AT1: overwrite; CMM (5) AT2: launch error AT3: CMM (7)
	126	1	1,2	1	1	1	1	9	AT3: CMM (7)
	127					2	16	23	ABEND: AT1
	128	2	5					32	AT1: CMM (7) AT3: CMM (12)
	129	3	3			3	12	90	AT3: CMM (13)
	130					4	2	135	AT3: CMM (5,9,10,11 14,15)
	131					5	6,9,10,11 14,15	150	AT3: (4,8,13)
	132					6	5,8,13	160	AT1: CMM (8,13)
	133	4	6					176	AT1: CMM (3)
	134	5	7					227	AT1: CMM (5)
	135	6	4					892	ABEND: AT1
	136	7	8					2351	ABEND: AT3
	137					7	18	4201	ABEND: AT3
	138					8	17	10000	End of Rep.
3	139	0	0	0	0	0	0	0	AT1: overwrite, CMM (5)
	140	1	1,2					2	AT2: launch error AT3: CMM (7)
	141			1	1	1	1	21	AT3: (5,9,10,11,12 13,14,15)
	142					2	2,6,9,10,11, 12,14,15	68	AT1: CMM (7)
	143	2	3					77	ABEND: AT1
	144	3	5					94	AT3: CMM (7)
	145					3	16	141	AT3: CMM (4,8,13)
	146					4	5,8,13	279	AT1: CMM (8,13)
	147	4	6					370	AT1: CMM (3)
	148	5	7					539	AT1: CMM (5)
	149	6	4					568	ABEND: AT3
	150					5	18	2889	ABEND: AT3
	151					6	17	4968	ABEND: AT1
	152	7	8					10000	End of Rep.

		-AT1-		-AT2-		-AT3-			
Rep. No.	Seq. No.	D.S.	Fix Nos.	D.S.	Fix Nos.	D.S.	Fix Nos.	CASE	ERROR
13	314	0	0	0	0	0	0	0	AT1: overwrite AT3: CMM (7)
	315	1	1			1	1	1	AT1: CMM (5) AT2: launch error
	316	2	2	1	1			4	ABEND: AT1
	317	3	5					20	AT1: CMM (7)
	318	4	3					41	AT1: CMM (10) AT3: CMM (12,13)
	319	5	7			2	2,12	53	AT3: CMM (7)
	320					3	16	57	AT3: CMM (4,5,8,9,10 11,13,14,15)
	321					4	5,6,8,9,10 11,13,14,15	86	AT1: CMM (8,13)
	322	6	6					318	AT1: CMM (5)
	323	7	4					3970	ABEND: AT3
	324					5	18	4159	AT1: CMM (10) ABEND: AT3
	325	8	11			6	17	4159	AT3: CMM (10)
	326					7	19	6605	ABEND: AT1
	327	9	8					10000	End of Rep.
14	328	0	0	0	0	0	0	0	AT1: overwrite, CMM (5) AT2: launch error AT3: CMM (7)
	329	1	1,2	1	1	1	1	2	AT1: CMM (7)
	330	2	3					38	AT3: CMM (12,13)
	331					2	2,12	60	ABEND: AT1
	332	3	5					110	AT1: CMM (8,13) AT3: CMM (7)
	333	4	6			3	16	117	AT3: CMM (5,9,10,11 14,15)
	334					4	6,9,10,11, 14,15	133	AT3: CMM (4,8,13)
	335					5	5,8,13	149	AT1: CMM (3)
	336	5	7					237	ABEND: AT3
	337					6	18	1077	AT1: CMM (5)
	338	6	4					1899	ABEND: AT1
	339	7	8					10000	End of Rep.
15	340	0	0	0	0	0	0	0	AT1: overwrite AT2: launch error AT3: CMM (7)
	341	1	1	1	1	1	1	1	AT1: CMM (5)
	342	2	2					3	AT1: CMM (5) AT3: CMM (4,6,8,9,10, 11,12,13,14,15)
	343	3	4			2	5,6,8,9,10 11,12,13, 14,15	13	ABEND: AT1
	344	4	5					16	AT1: CMM (7)
	345	5	3					23	AT1: CMM (8,13)
	346	6	6					38	AT3: CMM (7)
	347					3	16	316	AT1: CMM (10)
	348	7	7					507	ABEND: AT3
	349					4	17	844	ABEND: AT3
	350					5	18	5234	ABEND: AT1
	351	8	8					10000	End. of Rep.

		-AT1-		-AT2-		-AT3-			
Rep. No.	Seq. No.	D.S.	Fix Nos.	D.S.	Fix Nos.	D.S.	Fix Nos.	CASE	ERROR
16	352	0	0	0	0	0	0	0	AT1: overwrite
	353	1	1					1	AT1: CMM (5)
	354	2	2					3	ABEND: AT1
	355	3	5					5	AT2: launch error AT3: CMM (7)
	356			1	1	1	1	7	AT1: CMM (10) AT3: CMM (12,13)
	357	4	7			2	2,12	12	AT1: CMM (5) AT3: CMM (5,9,10,11, 14,15)
	358	5	4			3	6,9,10,11, 14,15	31	AT1: CMM (7)
	359	6	3					116	AT1: CMM (8,13)
	360	7	6					173	AT3: CMM (7)
	361					4	16	182	AT3: CMM (4,8,13)
	362					5	5,8,13	183	ABEND: AT1
	363	8	8					1322	ABEND: AT3
	364					6	18	2137	ABEND: AT3
	365					7	17	10000	End of Rep.
17	366	0	0	0	0	0	0	0	AT1: overwrite, CMM (5) AT2: launch error AT3: CMM (7)
	367	1	1,2	1	1	1	1	10	AT3: CMM (13)
	368					2	2	19	ABEND: AT1
	369	2	7					29	AT1: CMM (7)
	370	3	3					34	AT3: CMM (12)
	371					3	12	35	AT3: CMM (4,5,8,9,10, 11,13,14,15)
	372					4	5,6,8,9,10, 11,13,14,15	49	AT1: CMM (8,13)
	373	4	6					78	AT3: CMM (7)
	374					5	16	372	ABEND: AT1
	375	5	5					453	AT1: CMM (5)
	376	6	4					836	ABEND: AT3
	377					6	18	4812	ABEND: AT1
	378	7	8					6634	ABEND: AT3
	379					7	17	10000	End of Rep.
18	380	0	0	0	0	0	0	0	AT1: overwrite, CMM (5) AT2: launch error AT3: CMM (7)
	381	1	1,2	1	1	1	1	14	AT3: CMM (12,13)
	382					2	2,12	15	ABEND: AT1
	383	2	7					43	AT3: CMM (7)
	384					3	16	80	ABEND: AT1
	385	3	5					112	AT1: CMM (8,13)
	386	4	6					133	AT3: CMM (5,9,10,11, 14,15)
	387					4	6,9,10,11, 14,15	166	AT1: CMM (7)
	388	5	3					323	AT3: CMM (4,8,13)
	389					5	5,8,13	979	AT1: CMM (5)
	390	6	4					1142	ABEND: AT3
	391					6	18	4401	ABEND: AT3
	392					7	17	10000	End of Rep.

		-AT1-		-AT2-		-AT3-			
Rep. No.	Seq. No.	D.S.	Fix Nos.	D.S.	Fix Nos.	D.S.	Fix Nos.	CASE	ERROR
25	628	0	0	0	0	0	0	0	AT1: Overwrite AT3: CMM (12,13) = 0
	629	1	1			1	2,12	2	ABEND: AT1 RAD line 31 AT1: CMM (5)
	630	2	5					2	AT2: launch error AT3: CMM (7) = 0
	631	3	2	1	1	2	1	7	AT1: CMM (7) = 1
	632	3	3					20	AT3: CMM (4,5,8,9, 10,11,13,14,15) = 1
	633					3	5,6,8,9 10,11,13 14,15	36	AT1: CMM (8,13) = 1
	634	4	6					103	AT3: CMM (7) = 0
	635					4	16	110	AT1: CMM (5) = 1
	636	5	4					688	AT1: CMM (3) = 0
	637	6	7					978	ABEND: AT1 RADCIR line 58
	638							4474	ABEND: AT1 ANGLEA line 27
	639	7	8					5553	ABEND: AT3 AGLCOS line 26
	640					6	17	8358	AT1: CMM (1) = 0
	641	8	9						

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-177930		

4. Title and Subtitle	5. Report Date
AUTOSIM: An Automated Repetitive Software Testing Tool	September 1985
	6. Performing Organization Code
7. Author(s) J. R. Dunham, S. E. McBride	8. Performing Organization Report No.
	10. Work Unit No.
9. Performing Organization Name and Address Research Triangle Institute Research Triangle Park, NC 27709	11. Contract or Grant No. NAS1-16489
	13. Type of Report and Period Covered Contractor Report
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546	14. Sponsoring Agency Code 505-34-13-32

15. Supplementary Notes

Langley Technical Monitor: Gerard E. Migneault

16. Abstract

AUTOSIM is a software tool which automates the repetitive run testing of software. This tool executes programming tasks previously performed by a programmer with one year of programming experience. Use of the AUTOSIM tool requires a knowledge base containing information about known faults, code fixes, and the fault diagnosis-correction process. AUTOSIM can be considered as an "expert" system which replaces a low level of programming expertise.

The report contains reference information about the design and implementation of the AUTOSIM software test tool, provides flowcharts to assist in maintaining the software code, and documents how to use the tool.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Software reliability Software error rates	Unclassified - Unlimited Subject Category 61

19. Security Classif. (of this report)	20. Security Classif (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	97	

For sale by the National Technical Information Service, Springfield, Virginia 22161

End of Document